Triangulating the Square and Squaring the Triangle: Quadtrees and Delaunay Triangulations are Equivalent

Maarten Löffler[*] Wolfgang Mulzer[†]

Abstract

We show that Delaunay triangulations and compressed quadtrees are equivalent structures. More precisely, we give two algorithms: the first computes a compressed quadtree for a planar point set, given the Delaunay triangulation; the second finds the Delaunay triangulation, given a compressed quadtree. Both algorithms run in deterministic linear time on a pointer machine. Our work builds on and extends previous results by Krznaric and Levcopolous [40] and Buchin and Mulzer [10]. Our main tool for the second algorithm is the well-separated pair decomposition (WSPD) [13], a structure that has been used previously to find Euclidean minimum spanning trees in higher dimensions [27]. We show that knowing the WSPD (and a quadtree) suffices to compute a planar EMST in *linear* time. With the EMST at hand, we can find the Delaunay triangulation in linear time [21].

As a corollary, we obtain deterministic versions of many previous algorithms related to Delaunay triangulations, such as splitting planar Delaunay triangulations [19, 20], preprocessing imprecise points for faster Delaunay computation [9, 42], and transdichotomous Delaunay triangulations [10, 15, 16].

1 Introduction

Delaunay triangulations and quadtrees are among the oldest and best-studied notions in computational geometry [4,7,25,29,44,45,47,49], captivating the attention of researchers for almost four decades. Both are proximity structures on planar point sets; Figure 1 shows a simple example of these structures. Here, we will demonstrate that they are, in fact, equivalent in a very strong sense. Specifically, we describe two algorithms. The first computes a suitable quadtree for P, given the Delaunay triangulation $\mathrm{DT}(P)$. This algorithm closely follows a previous result by Krznaric and Levcopolous [40], who solve this problem in a stronger model of computation. Our contribution lies in adapting their algorithm

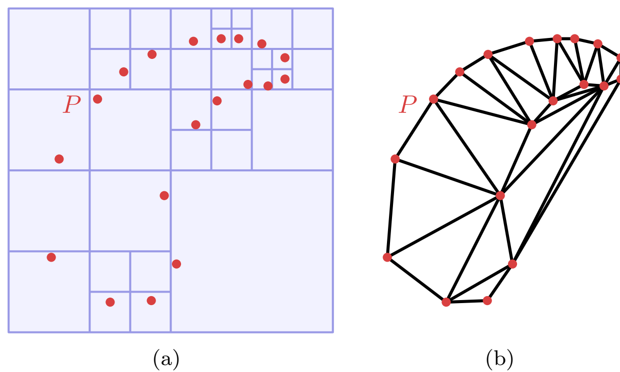

Figure 1: A planar point set P, and a quadtree (a) and a Delaunay triangulation (b) on it.

to the real RAM/pointer machine model.[1] The second algorithm, which is the main focus of this paper, goes in the other direction and computes $\mathrm{DT}(P)$, assuming that a suitable quadtree for P is at hand. This connection was first discovered and fruitfully applied by Buchin and Mulzer [10] (see also [9]). While their approach is to use a hierarchy of quadtrees for faster conflict location in a randomized incremental construction of $\mathrm{DT}(P)$, we pursue a strategy similar to the one by Löffler and Snoeyink [42]: we use the additional information to find a connected subgraph of $\mathrm{DT}(P)$, from which $\mathrm{DT}(P)$ can be computed in linear deterministic time [21]. As in Löffler and Snoeyink [42], our subgraph of choice is the *Euclidean minimum spanning tree* (EMST) for P, $\mathrm{emst}(P)$ [27]. The connection between quadtrees and EMSTs is well known: several algorithms use the *well-separated pair decomposition* (WSPD) [13], or a variant thereof, to reduce EMST computation to solving the *bichromatic closest pair* problem. In that problem, we are given two point sets R and B, and we look for a pair $(r, b) \in R \times B$ that minimizes the distance $|rb|$ [1, 12, 41, 52]. Given a quadtree for P, a WSPD for P can be found in linear time [9, 13, 14, 35]. EMST algorithms based on bichromatic closest pairs constitute the

[*]Computer Science Department, University of California, Irvine; Irvine, CA 92697, USA; *mloffler@uci.edu*.

[†]Institut für Informatik, Freie Universität Berlin; 14195 Berlin, Germany; *mulzer@inf.fu-berlin.de*.

[1]Refer to Appendix A for a description of different computational models.

fastest known solutions in higher dimensions. Our approach is quite similar, but we focus exclusively on the plane. We use the quadtree and WSPDs to obtain a sequence of bichromatic closest pair problems, which then yield a sparse supergraph of the EMST. There are several issues: we need to ensure that the bichromatic closest pair problems have total linear size and can be solved in linear time, and we also need to extract the EMST from the supergraph in linear time. In Section 4, we show how to do this using the structure of the quadtree, combined with a partition of the point set according to angular segments similar to Yao's technique [52].

1.1 Applications. Our two algorithms have several implications for derandomizing recent algorithms related to DTs. First, we mention *hereditary* computation of DTs. Chazelle *et al.* [19] show how to *split* a Delaunay triangulation in linear expected time (see also [20]). That is, given $\mathrm{DT}(P \cup Q)$, they describe a randomized algorithm to find $\mathrm{DT}(P)$ and $\mathrm{DT}(Q)$ in expected time $O(|P| + |Q|)$. Knowing that DTs and quadtrees are equivalent, this result becomes almost obvious, as quadtrees are easily split in linear time. More importantly, our new algorithm achieves linear *worst-case* running time. Clarkson and Seshadhri [22] use hereditary DTs for *self-improving algorithms* [2, 3]. Together with the ε-net construction by Pyrga and Ray [46] (see [2, Appendix A]), our result yields a deterministic version of their algorithm for point sets generated by a random source (the inputs are probabilistic, but not the algorithm).

Eppstein *et al.* [28] introduce the skip-quadtree and show how to turn a (compressed) quadtree into a skip-quadtree in linear time. Buchin and Mulzer [10] use a (randomized) skip-quadtree to find the DT in linear expected time. This yields several improved results about computing DTs. Most notably, they show that in the *transdichotomous* setting [15, 16, 30], computing DTs is no harder than sorting the points (according to some special order). Here, we show how to go directly from a quadtree to a DT, without skip-quadtrees or randomness. This gives the first *deterministic* transdichotomous reduction from DTs to sorting.

Buchin *et al.* [9] use both hereditary DTs and the connection between skip-quadtrees and DTs to simplify and generalize an algorithm by Löffler and Snoeyink [42] to preprocess imprecise points for Delaunay triangulation in linear expected time (see also Devillers [26] for another simplified solution). Löffler and Snoeyink's original algorithm is deterministic, and the derandomized version of the Buchin *et al.* algorithm proceeds in a very similar spirit. However, we now have an optimal deterministic solution for the generalized problem as well.

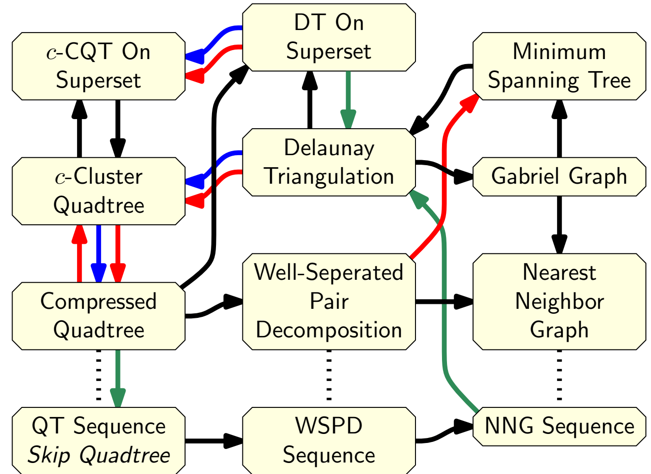

Figure 2: We show which can be computed from which in linear time. The black arrows depict known linear time deterministic algorithms that work in the pointer machine/real RAM model. The red arrows depict our new results. Furthermore, for reference, we also show known randomized linear time algorithms (in green) and known deterministic linear time algorithms that work in a weaker model of computation (in blue).

In Figure 2, we show a graphical representation of different proximity structures on planar point sets. The arrows show which structures can be computed from which in linear deterministic time on a pointer machine, before and after this paper. Please realize that there are several subtleties of different algorithms and their interactions that are hard to show in a diagram, it is included purely as illustration of the impact of our results.

2 Preliminaries

We review some known definitions, structures, and algorithms and their relations.

2.1 Delaunay Triangulations and Euclidean Minimum Spanning Trees. Given a set P of n points in the plane, an important and extensively studied structure is the *Delaunay triangulation* of P [4, 7, 25, 45, 49], denoted $\mathrm{DT}(P)$. It can be defined as the dual graph of the Voronoi diagram, the triangulation that optimizes the smallest angle in any triangle, or in many other equivalent ways, and it has been proven to optimize many other different criteria [44].

The *Euclidean minimum spanning tree* of P, denoted $\mathrm{emst}(P)$, is the tree of smallest total edge length that has the points of P as its vertices, and it is well known that the EMST is a subgraph of the DT [49, The-

orem 7]. In the following, we will assume that all the pairwise distances in P are distinct (a general position assumption), which implies that $\text{emst}(P)$ is uniquely determined. Finally, we remind the reader that $\text{emst}(P)$, like every minimum spanning try, has the following *cut property*: let $P = R \cup B$ a partition of P, and let r and b be the two points with $r \in R$ and $b \in B$ that minimize the distance $|rb|$. Then rb is an edge of $\text{emst}(P)$.[2]

2.2 Quadtrees—Compressed and c-Cluster. Let P be a planar point set. A *quadtree* for P is a hierarchical decomposition of an axis-aligned bounding square for P into smaller axis-aligned *squares* [4,29,34,47]. A *regular* quadtree is constructed by successively subdividing every square with at least two points into four congruent child squares. A node v of a quadtree stores (i) S_v, the square corresponding to v; (ii) P_v, the points contained in S_v (P_v is only stored implicitly); and (iii) B_v, the axis-aligned bounding box for P_v. We write $|S_v|$ and $|B_v|$ for the diameter of S_v and B_v, and c_v for the center of S_v. Furthermore, we denote the parent of v by $\bar{v}$. Regular quadtrees can have unbounded depth (if P has unbounded spread[3]), so in order to give any theoretical guarantees the concept is usually refined. In the sequel, we use two such variants of quadtrees, namely *compressed* and *c-cluster* quadtrees, which we show are in fact equivalent.

A *compressed* quadtree is a quadtree in which we replace long paths of nodes with only one child by a single edge [5,6,9]. It has size $O(|P|)$. Formally, given a large constant a, an a-compressed quadtree is a regular quadtree with additional *compressed* nodes.[4] A compressed node v has only one child $\tilde{v}$ with $|S_{\tilde{v}}| \leq |S_v|/a$ and such that $S_v \setminus S_{\tilde{v}}$ has no points from P. Note that $S_{\tilde{v}}$ need not be aligned with S_v, which would happen if we literally "compressed" a regular quadtree. This relaxed definition is necessary because existing algorithms for computing aligned compressed quadtrees use a more powerful model of computation than our real RAM/pointer machine (see Appendix A). This is usually acceptable for quadtrees,[5] but we intend to derandomize algorithms that work on a traditional real RAM/pointer machine, so we prefer to stay in this model. This keeps our results comparable with the previous work.

Now let c be a large enough constant. A subset $U \subseteq P$ is a *c-cluster* if $U = P$ or $d(U, P \setminus U) \geq c|B_U|$, where B_U denotes the smallest axis-aligned bounding box for U, and $d(A, B)$ is the minimum distance between a point in A and a point in B [39,40].[6] The c-clusters for P form a laminar family,[7] so they define a *c-cluster tree* T_c. These trees are a very natural way to tackle point sets of unbounded spread, and they have linear size. However, they also may have high degree. To avoid this, a c-cluster tree T_c can be augmented by additional nodes, adding more structure to the parts of the point set that are not strongly clustered. For each internal node u of T_c with set of children V, we build a regular quadtree on a set of points containing one representative point from each node in V (the intuition being that such a cluster is so small and far from its neighbors, that we might as well treat it as a point). This quadtree has size $O(|V|)$ (Lemma B.3), so we obtain a tree of constant degree and linear size, the *c-cluster quadtree*. The sets P_v, S_v and B_v for the c-cluster quadtree are just as for regular and compressed quadtrees, where in P_v we expand the representative points appropriately. Note that it is possible that $S_v \not\supseteq P_v$, but the points of P_v can never be too far from S_v. In Appendix B.1 we elaborate more on c-cluster quadtrees and their properties, and in Appendix B.2, we prove the following theorem:

THEOREM 2.1. *Let P be a planar point set. Given a c-cluster quadtree on P, we can compute in linear time an $O(c)$-compressed quadtree on P; and given an a-compressed quadtree on P, we can compute in linear time an $O(a)$-cluster quadtree on P.*

2.3 Well-Separated Pair Decompositions. For any two finite sets U and V, let $U \otimes V := \{\{u, v\} \mid u \in U, v \in V, u \neq v\}$. A *pair decomposition* $\mathcal{P}$ for a planar[8] n-point set P is a set of m *pairs* $\{\{U_1, V_1\}, \ldots, \{U_m, V_m\}\}$, such that (i) for all $i = 1, \ldots, m$, we have $U_i, V_i \subseteq P$ and $U_i \cap V_i = \emptyset$; and (ii) for any $\{p, q\} \in P \otimes P$, there is exactly one i with $\{p, q\} \in U_i \otimes V_i$. We call m the *size* of $\mathcal{P}$. Fix a constant $\varepsilon \in (0, 1)$, and let $\{U, V\} \in \mathcal{P}$. Denote by B_U, B_V the smallest axis-aligned rectangles containing U and V. We say that $\{U, V\}$ is *ε-well-separated* if $\max\{|B_U|, |B_V|\} \leq$

[2] This is very similar to the bichromatic closest pair reduction mentioned in the introduction, but note that the cut property holds for any partition of P, whereas the bichromatic closest pair reduction requires a very specific decomposition of P into pairs of subsets (which is usually not a partition).

[3] The *spread* of a point set P is the ratio between the largest and the smallest distance between any two distinct points in P.

[4] Such nodes are often called *cluster*-nodes in the literature [5,6,9], but we prefer the term *compressed* to avoid confusion with c-cluster quadtrees defined below.

[5] Indeed, as pointed out by Har-Peled [35, Lemma 2.2.6], some non-standard operation is *inevitable* if we require that the squares of the compressed quadtree are perfectly aligned.

[6] That is, U is a c-cluster precisely if $\{U, P \setminus U\}$ is a $(1/c)$-*semi-separated* pair [35,51].

[7] A *laminar family* is a set system in which any two sets A and B satisfy either $A \cap B = \emptyset$; $A \subseteq B$; or $B \subseteq A$.

[8] Although some of these notions extend naturally to higher dimensions, the focus of this paper is on the plane.

$\varepsilon d(U,V)$, where $d(U,V)$ is the distance between B_U and B_V (i.e., the smallest distance between a point in B_U and a point in B_V). If $\{U,V\}$ is not ε-well-separated, we say it is *ε-ill-separated.* We call $\mathcal{P}$ an *ε-well-separated pair decomposition* (ε-WSPD) if all its pairs are ε-well-separated [12,13,27,35].

Now let T be a (compressed or c-cluster) quadtree for P. Given $\varepsilon > 0$, it is well known that T can be used to obtain an ε-WSPD for P in linear time [13,35]. Since we will need some specific properties of such an ε-WSPD, we give pseudo-code for such an algorithm in Algorithm 1. The correctness of `wspd` is immediate, since it only outputs well-separated pairs, and the bounds on the running time and the size of $\texttt{wspd}(T)$ follow from a well-known volume argument which we omit [9,13,14,35].

Algorithm 1 Finding a well-separated pair decomposition.

1. Call $\texttt{wspd}(r)$ on the root r of T.

$\texttt{wspd}(v)$

1. If v is a leaf, return $\emptyset$.
2. Return the union of $\texttt{wspd}(w)$ and $\texttt{wspd}(\{w_1,w_2\})$ for all children w and pairs of distinct children w_1, w_2 of v.

$\texttt{wspd}(\{u,v\})$

1. If S_u and S_v are ε-well-separated, return $\{u,v\}$.
2. Otherwise, if $|S_u| \leq |S_v|$, return the union of $\texttt{wspd}(\{u,w\})$ for all children w of v.
3. Otherwise, return the union of $\texttt{wspd}(\{w,v\})$ for all children w of u.

Theorem 2.2. *There is an algorithm* `wspd`*, that given a (compressed or c-cluster) quadtree T for a planar n-point set P, finds in time $O(n)$ a linear-size ε-WSPD for P, denoted $\texttt{wspd}(T)$.* □

Note that the WSPD is not stored explicitly: we cannot afford to store all the pairs $\{U,V\}$, since their total size might well be quadratic. Instead, $\texttt{wspd}(T)$ contains pairs $\{u,v\}$, where u and v are nodes in T, and $\{u,v\}$ is used to represent the pair $\{P_u, P_v\}$.

Note that the algorithm computes the WSPD with respect to the squares S_v, instead of the bounding boxes B_v. This makes no big difference, since for compressed quadtrees $B_v \subseteq S_v$, and for c-cluster quadtrees S_v can be outside B_v only for c-cluster nodes, resulting in a loss of at most a factor $1+1/c$ in separation. Referring to the pseudo-code in Algorithm 1, we now prove three observations.

Observation 2.3. *Let $\{u,v\}$ be a pair of distinct nodes of T. If $\texttt{wspd}(\{u,v\})$ is executed by* `wspd` *run on T (in particular, if $\{u,v\} \in \texttt{wspd}(T)$), then $\max\{|S_u|,|S_v|\} \leq \min\{|S_{\bar u}|,|S_{\bar v}|\}$.*

Proof. We use induction on the depth of the call stack for $\texttt{wspd}(\{u,v\})$. Initially, u and v are children of the same node, and the statement holds. Furthermore, assuming that $\texttt{wspd}(\{u,v\})$ is called by $\texttt{wspd}(\{u,\bar v\})$ (and hence $|S_u| \leq |S_{\bar v}|$), we get $\max\{|S_u|,|S_v|\} \leq |S_{\bar v}| = \min\{|S_{\bar u}|,|S_{\bar v}|\}$, where the last equation follows by induction. □

Observation 2.4. *If $\{u,v\} \in \texttt{wspd}(T)$, then $\bar u$ and $\bar v$ are ill-separated.*

Proof. If $\bar u = \bar v$, then the claim is obvious. Otherwise, let us assume that $\texttt{wspd}(\{u,v\})$ was called by $\texttt{wspd}(\{u,\bar v\})$. This means that $\{u,\bar v\}$ is ill-separated and $\max\{|S_u|,|S_{\bar v}|\} = |S_{\bar v}|$. Therefore, $\max\{|S_{\bar u}|,|S_{\bar v}|\} \geq |S_{\bar v}| > \varepsilon d(u,\bar v) \geq \varepsilon d(\bar u,\bar v)$, and $\{\bar u,\bar v\}$ is ill-separated. □

Claim 2.5. *Let $\{u,v\} \in \texttt{wspd}(T)$. Then there exist squares R_u and R_v such that (i) $S_u \subseteq R_u \subseteq S_{\overline{u}}$ and $S_v \subseteq R_v \subseteq S_{\overline{v}}$; (ii) $|R_u| = |R_v|$; and (iii) $|R_u|/2\varepsilon \leq d(R_u,R_v) \leq 2|R_u|/\varepsilon$.*

Proof. Suppose $\texttt{wspd}(\{u,v\})$ is called by $\texttt{wspd}(\{u,\overline{v}\})$, the other case is symmetric. Let us define $r := \min\{\varepsilon d(u,v), |S_{\overline{v}}|\}$. By Observation 2.3, we have $|S_u|,|S_v| \leq |S_{\overline{v}}| \leq |S_{\overline{u}}|$. Since $\{u,v\}$ is well-separated, we have $\varepsilon d(u,v) \geq \max\{|S_u|,|S_v|\}$. Hence, $|S_{\overline{u}}|,|S_{\overline{v}}| \geq r \geq |S_u|,|S_v|$, and we can pick squares R_u and R_v of diameter r that fulfill (i). Now (ii) holds by construction, and it remains to check (iii). First, note that $d(R_u,R_v) \geq d(u,v) - 2r \geq (1-2\varepsilon)d(u,v) \geq r/2\varepsilon$, for $\varepsilon \leq 1/4$. This proves the lower bound. For the upper bound, observe that $\varepsilon d(u,v) \leq \varepsilon(d(u,\overline{v}) + |S_{\overline{v}}|) \leq (1+\varepsilon)|S_{\overline{v}}|$, because $\{u,\overline{v}\}$ is ill-separated. Thus, we have $r \geq \varepsilon d(u,v)/2$, and $d(R_u,R_v) \leq d(u,v) \leq 2r/\varepsilon$, as desired. □

3 From Delaunay Triangulations to c-Cluster Quadtrees

For the first direction of our equivalence we need to show how to compute a c-cluster quadtree for P when given $\mathrm{DT}(P)$. This was already done by Krznaric and Levcopolous [39,40], but their algorithm works in a stronger model of computation which includes the

floor function and allows access to data at the bit level. As argued above, we prefer the real RAM/pointer machine, so we need to do some work to adapt their algorithm to our computational model. In Appendix C, we describe how Krznaric and Levcopolous's algorithm can be modified to avoid bucketing and bit-twiddling techniques. The only difference is that in the resulting c-cluster quadtree the squares for the c-clusters are not perfectly aligned with the squares of the parent quadtree. In our setting, this does not matter.

THEOREM 3.1. *Given* $\mathrm{DT}(P)$*, we can compute a* c*-cluster quadtree for* P *in linear deterministic time on a pointer machine.*

4 From a c-Cluster Quadtree to the Delaunay Triangulation

We now go in the opposite direction, and show how to construct a DT from a WSPD. Let P be a set of points, and T a compressed quadtree for P. Throughout this section, ε is a small enough constant (say, $\varepsilon = \pi/400$), and k is a large enough constant (e.g., $k = 100$). Let u and v be two *unrelated* nodes of T, i.e., neither node is an ancestor of the other. Let L_{uv} be the set of directed lines that stab S_u before S_v. The set $\Phi_{uv} \subseteq [0, 2\pi)$ of directions for L_{uv} is an interval modulo 2π whose extreme points correspond to the two diagonal bitangents of S_u and S_v, i.e., the two lines that meet S_u and S_v in exactly one point each and have S_u and S_v to different sides.

OBSERVATION 4.1. *Let* u *and* v *be two unrelated nodes of* T*, and let* $\tilde{u}$ *be a descendant of* u *and* $\tilde{v}$ *be a descendant of* v*. Then* $\Phi_{\tilde{u}\tilde{v}} \subseteq \Phi_{uv}$*.*

Proof. This is clear, because $S_{\tilde{u}} \subseteq S_u$ and $S_{\tilde{v}} \subseteq S_v$. □

OBSERVATION 4.2. *If* u *and* v *are two nodes of* T *such that* $\{u, v\}$ *is* ε*-well-separated, then* $|\Phi_{uv}| \leq 8\varepsilon$*.*

Proof. Let $d := |c_u c_v|$, D_u be the disk around c_u with radius εd, and D_v the disk around c_v with the same radius. By well-separation, $S_u \subseteq D_u$ and $S_v \subseteq D_v$. Let β be the angle between the diagonal bitangents of D_u and D_v. Then $|\Phi_{uv}| \leq \beta$, and $\beta = 2\arcsin(\varepsilon d/\frac{1}{2}d) = 2\arcsin(2\varepsilon) \leq 8\varepsilon$, as claimed. □

For a direction $\phi \in [0, 2\pi[$ we define $\Phi_\phi := \{\psi \bmod 2\pi \mid \psi \in [\phi - \varepsilon/2, \phi + \varepsilon/2]\}$, i.e., the set of all directions that differ from ϕ by at most $\varepsilon/2$. We say that an ordered pair (u, v) has direction ϕ if $\Phi_{uv} \cap \Phi_\phi \neq \emptyset$. We also say that a pair of points (p, q) has direction ϕ if the corresponding pair in the WSPD has direction ϕ. For a given point p in the plane, we define the ε-cone $\mathcal{C}_\phi(p)$ as the cone with apex p and opening angle ε centered around ϕ.

4.1 Constructing a Supergraph of the EMST.

In the following, we abbreviate $\mathcal{P} := \mathtt{wspd}(T)$. The goal of this section is to construct a graph H with vertex set P and $O(n)$ edges, such that $\mathrm{emst}(P) \subseteq H$. It is well known that if we take the graph H' on P with edge set $E := \{e_{uv} \mid \{u, v\} \in \mathcal{P}\}$, where each e_{uv} connects the bichromatic closest pair for P_u and P_v, then H' contains $\mathrm{emst}(P)$ and has $O(n)$ edges [27]. However, as defined, it is not clear how to find H' in linear time. There are several major obstacles. Firstly, even though the tree T has $O(n)$ vertices, it may well be that $\sum_{u \in T} |P_u| = \Omega(n^2)$. Secondly, even if the total size of all P_u's was $O(n)$, we still need to find bichromatic closest pairs for all *pairs* in $\mathcal{P}$. Thus, it could happen that a large set P_u appears in many pairs of $\mathcal{P}$, making the total problem size superlinear. Thirdly, we need to actually solve the bichromatic closest pair problems. A straightforward solution to find the bichromatic closest pair for sets R and B with sizes r and b would take time $O((r + b)\log(\min(r, b)))$, by computing the Voronoi diagram for the smaller set and locating all points from the other set in it. We need to find a way to do it in linear time.

To address these problems, we actually construct a slightly larger graph H, by partitioning the pairs in $\mathcal{P}$ according to their orientation. More precisely, let $Y = \{0, \varepsilon, 2\varepsilon, \ldots, (l - 1)\varepsilon\}$ be a set of l directions, where we assume that $l = 2\pi/\varepsilon$ is an integer. For every direction $\phi \in Y$, we construct a graph H_ϕ with $O(n)$ edges and then let $H = \bigcup_{\phi \in Y} H_\phi$. Given $\phi \in Y$, the graph H_ϕ is constructed in three steps:

1. For every node $u \in T$, select a subset $Z_u \subseteq P_u$, such that $\sum_{u \in T} |Z_u| = O(n)$, and such that $\{\{p, q\} \mid p \in Z_u, q \in Z_v, \{u, v\} \in \mathcal{P}\}$ still contains all edges of $\mathrm{emst}(P)$ with orientation ϕ. This addresses the first problem by making the total set size linear.

2. Find a subset $\mathcal{P}' \subseteq \mathcal{P}$, such that each $u \in T$ appears in $O(1)$ pairs of $\mathcal{P}'$, and the set $\{\{p, q\} \mid p \in Z_u, q \in Z_v, \{u, v\} \in \mathcal{P}'\}$ contains all edges of $\mathrm{emst}(P)$ with orientation ϕ. In particular, we choose for every node $u \in T$ a subset $\mathcal{P}_u \subseteq \mathcal{P}$ such that $\mathcal{P}' = \bigcup_{u \in T} \mathcal{P}_u$, each pair in $\mathcal{P}_u$ contains u, and $|\mathcal{P}_u| = O(1)$. This addresses the second problem by ensuring that every set appears in $O(1)$ pairs.

3. For every pair $\{u, v\} \in \mathcal{P}'$, we include in H_ϕ the edge pq such that $\{p, q\}$ is the closest pair in $Z_u \otimes Z_v$ (i.e., $\{p, q\} = \mathrm{argmin}_{\{p', q'\} \in Z_u \otimes Z_v} |p'q'|$). Here we actually solve all the bichromatic closest pair problems.

Clearly, H_ϕ has $O(n)$ edges, and we will show that H is indeed a supergraph of $\mathrm{emst}(P)$. Our strategy of

subdividing the edges according to their orientation goes back to Yao, who used a similar scheme to find EMSTs in higher dimensions [52].

Step 1: Finding the Z_u's. Recall that we fixed a direction $\phi \in Y$. Take the set $\mathcal{P}_\phi \subseteq \texttt{wspd}(T)$ of pairs with direction ϕ. For a pair $\pi \in \mathcal{P}_\phi$, we write (u, v) for the tuple such that $\pi = \{u, v\}$ and c_u comes before c_v in direction ϕ, it is a *directed* pair in $\mathcal{P}_\phi$. Call a node u of T *full* if either (i) u is the root; (ii) u is a non-empty leaf; or (iii) $\mathcal{P}_\phi$ has a directed pair (u, v). Let T' be the tree obtained from T by connecting every full node to its closest full ancestor, and by removing the other nodes. We can compute T' in linear time through a post-order traversal. Now, for every leaf v of T', put the point $p \in P_v$ into the sets Z_u, where u is one the k closest ancestors of v in T'. Repeat this procedure, while changing property (iii) above so that $\mathcal{P}_\phi$ has a directed pair (v, u), and augment the Z_u appropriately. This takes linear time, and $\sum_{u \in T} |Z_u| = O(n)$. Intuitively, Z_u contains those points of P_u that are sufficiently on the outside of the point set in direction ϕ. Figure 4.1 shows an example. Variants of the following claim have appeared several times before [1, 52].

Claim 4.3. *Let $p \in P$, and let $\mathcal{C}^+_\phi(p)$ denote the cone with apex p and opening angle 17ε centered around ϕ. Suppose that pq is an edge of $\mathrm{emst}(P)$ and $q \in \mathcal{C}^+_\phi(p)$. Then q is the nearest neighbor of p in $\mathcal{C}^+_\phi(p) \cap P$.*

Proof. If pq is an edge of $\mathrm{emst}(P)$, then the *lune* L defined by p and q contains no point of P [4].[9] Since the opening angle of $\mathcal{C}^+_\phi(p)$ is at most $\pi/3$, for ε small enough, the intersection of $\mathcal{C}^+_\phi(p)$ with L equals the intersection of $\mathcal{C}^+_\phi(p)$ with the disk around p of radius $|pq|$. Hence, q must be the nearest neighbor of p in $\mathcal{C}^+_\phi(p) \cap P$. □

Lemma 4.4. *Let pq be an edge of $\mathrm{emst}(P)$ with direction ϕ, and let $\{u, v\}$ be the corresponding wspd-pair. Then $\{p, q\} \in Z_u \otimes Z_v$.*

Proof. Let w be the leaf for p, and suppose for contradiction that $p \notin Z_u$, i.e., u is not among the k closest ancestors of w in T'. This means there exists a chain[10] $u_1, u_2, \ldots, u_k, u$ of $k + 1$ distinct ancestors of w, such that there are well-separated pairs $\{u_1, v_1\}, \{u_2, v_2\}, \ldots, \{u_k, v_k\} \in \mathcal{P}_\phi$.

Let $\mathcal{C}^+_\phi(p)$ be the cone with apex p and opening angle 17ε centered around ϕ. By Observation 4.2, we have

[9] L is the intersection of two disks with radius $|pq|$, one centered at p, the other centered at q.

[10] By a *chain* we mean a sequence $v_1, v_2, \ldots$ of nodes such that v_i is a descendant of v_{i+1}, for all i.

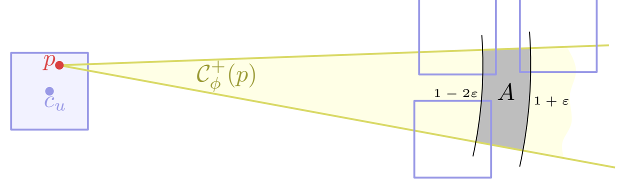

Figure 4: All squares R_w intersect the region A.

$S_v, S_{v_1}, \ldots, S_{v_k} \subseteq \mathcal{C}^+_\phi(p)$. Furthermore, since $\{u, v\}$ is well-separated, $d(u, v) \geq |S_u|/\varepsilon$. Now Claim 2.5 implies that there are squares R_{u_1}, R_{v_1} such that (i) $S_{u_1} \subseteq R_{u_1} \subseteq S_{u_2}$ and $S_{v_1} \subseteq R_{v_1}$; (ii) $|R_{u_1}| = |R_{v_1}|$; and (iii) $d(R_{u_1}, R_{v_1}) \leq 2|R_{u_1}|/\varepsilon$. This means that $d(p, P_{v_1}) \leq 2(1+1/\varepsilon)|R_{u_1}| \leq 2(1+1/\varepsilon)|S_{u_2}| \leq 2(1+1/\varepsilon)|S_u|/2^{k-1}$. Since $2(1 + 1/\varepsilon)/2^{k-1} < 1/\varepsilon$ for $k \geq 3$, this contradicts Claim 4.3 and the fact that $d(u, v) \geq |S_u|/\varepsilon$. A symmetric argument shows $q \in Z_v$. □

Step 2: Finding the $\mathcal{P}_u$'s. For every node $u \in T$, we include in $\mathcal{P}_u$ the k shortest pairs in direction ϕ, i.e., the pairs $\{u, v\} \in \texttt{wspd}(T)$ such that (i) c_v is contained in the ε-cone $\mathcal{C}_\phi(c_u)$ with apex c_u centered around direction ϕ; and (ii) there are less than k pairs $\{u, v'\} \in \texttt{wspd}(T)$ that fulfill (i) and have $|c_u c_{v'}| < |c_u c_v|$. Since k is constant, the $\mathcal{P}_u$'s can be constructed in total linear time. Even though each $\mathcal{P}_u$ contains a constant number of elements, a node might still appear in many such sets, so we further prune the pairs: by examining the $\mathcal{P}_u$'s, determine for each $v \in T$ the set $\mathcal{Q}_v = \{u \in T \mid v \in \mathcal{P}_u\}$. For each $\mathcal{Q}_v$, find the k closest neighbors of v in $\mathcal{Q}_v$, and for all other $\mathcal{P}_u$'s remove the corresponding pairs $\{u, v\}$. Now each node appears in only a constant number of pairs of $\mathcal{P}' = \bigcup_{u \in T} \mathcal{P}_u$.

Lemma 4.5. *Let pq be an edge of $\mathrm{emst}(P)$ with orientation ϕ, and let $\{u, v\}$ be the corresponding wspd-pair. Then $\{u, v\} \in \mathcal{P}_u$.*

Proof. We show that v is among the k closest neighbors of u in direction ϕ, a symmetric argument shows that u is among the k closest neighbors of v in direction $-\phi$. We may assume that $|c_u c_v| = 1$. Suppose that $\{u, v\}$ is not among the k shortest pairs in direction ϕ. Then there is a set W of k nodes of T such that for all $w \in W$ we have (i) $c_w \in \mathcal{C}_\phi(c_u)$; (ii) $|c_u c_w| < 1$; and (iii) $\{u, w\} \in \texttt{wspd}(T)$. By Claim 2.5, there exists for every $w \in W$ a pair of squares $R_u(w), R_w$ such that $S_u \subseteq R_u(w)$, $S_w \subseteq R_w$ and $|R_u(w)| = |R_w| \leq 2\varepsilon d(R_u(w), R_w) \leq 2\varepsilon$.

Let $\mathcal{C}^+_\phi(p)$ be the cone with apex p and opening angle 17ε centered around ϕ. By Observation 4.2, $S_w \subseteq \mathcal{C}^+_\phi(p)$ for all $w \in W$. Furthermore, every S_w contains a point at distance at most $1 + \varepsilon$ from p,

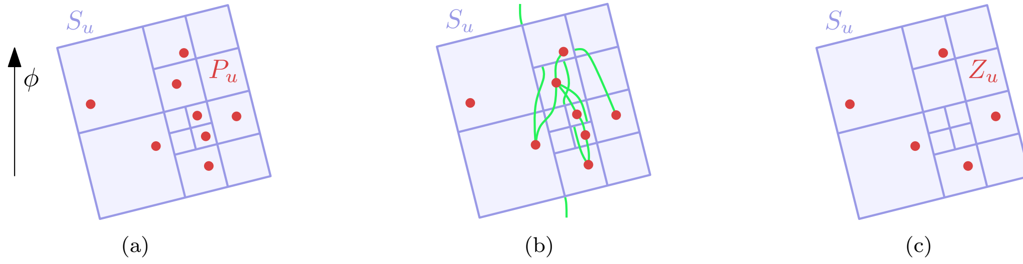

Figure 3: (a) A node u in the quadtree, with $|P_u| = 8$. (b) The relevant WSPD edges (in green) for the points in P_u in direction ϕ (up). There are also WSPD edges connecting u to other nodes above and below it. (c) For $k = 1$, Z_u contains those points $p \in P_u$ for which the smallest vertical WSPD pair involving p has u in the pair. In other words, they are the points that do not have a green edge in both directions in (b).

because $|c_w p| \le |c_w c_u| + |c_u p| \le 1 + \varepsilon$. Also, by Claim 4.3, every S_w contains a point at distance at least $|pq| \ge |c_u c_v| - |c_u p| - |q c_v| \ge 1 - 2\varepsilon$ from p. Thus, since $d(R_u(w), R_w) \le 2|R_w|/\varepsilon$ and $d(R_u(w), R_w) \ge 1 - 2\varepsilon - 2|R_w|$, we get $|R_w| \ge \varepsilon/8$, for ε small enough. However, this implies that W has only constantly many squares: all S_w (and hence all R_w) intersect the annular segment A inside $\mathcal{C}^+_\phi(p)$ with inner radius $1 - 2\varepsilon$ and outer radius $1 + \varepsilon$ (see Figure 4). All $w \in W$ are unrelated, since they are paired with u in $\texttt{wspd}(T)$. Furthermore, the set A has diameter $O(\varepsilon)$. If $w \in W$ is a compressed node, then R_w intersects no other $S_{w'}$, for $w' \in W$. Otherwise, $|S_w| \ge |R_w|/2$, and S_w is disjoint from all other $S_{w'}$. There can be only constantly many disjoint squares of size $\Omega(\varepsilon)$ meeting a region of diameter $O(\varepsilon)$, so choosing k large enough leads to a contradiction. □

Step 3: Finding the Nearest Neighbors. Unlike in the previous steps, the algorithm for Step 3 is a bit involved, so we switch the order and begin by showing correctness.

LEMMA 4.6. *Let pq be an edge of $\text{emst}(P)$ with direction ϕ and let $\{u, v\}$ be the corresponding wspd-pair. Then $\{p, q\}$ is the closest pair in $Z_u \otimes Z_v$.*

Proof. By Lemma 4.4, we have $\{p, q\} \in Z_u \otimes Z_v$. Furthermore, the cut property of minimum spanning trees implies that $pq \in \text{emst}(Z_u \cup Z_v)$. Since $\{u, v\}$ is well-separated, we have

$$(4.1) \qquad \max_{\{p',q'\} \in Z_u \otimes Z_u \cup Z_v \otimes Z_v} |p'q'| < \min_{\{p',q'\} \in Z_u \otimes Z_v} |p'q'|.$$

Now consider an execution of Kruskal's MST algorithm on $Z_u \cup Z_v$ [23, Chapter 23.2]. Let $\{p', q'\}$ be the closest pair in $Z_u \otimes Z_v$. By (4.1), the algorithm considers $p'q'$ only after processing all edges in $Z_u \otimes Z_u \cup Z_v \otimes Z_v$. Hence, at that point the sets Z_u and Z_v are each

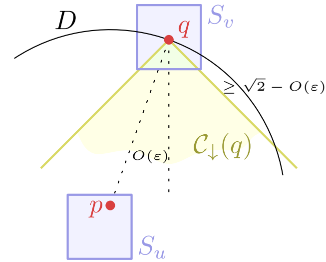

Figure 5: The intersection points of D and the boundary of $\mathcal{C}_\downarrow(q)$ lie outside S_v, so $S_v \cap \mathcal{C}_\downarrow(q) \subseteq D$.

contained in a connected component of the partial spanning tree, and $\text{emst}(Z_u \cup Z_v)$ can have at most one edge from $Z_u \otimes Z_v$. Hence, it follows that $\{p, q\} = \{p', q'\}$, as claimed. □

We now describe the algorithm. For ease of exposition, we take $\phi = \pi/2$ (i.e., we assume that P is rotated so that ϕ points in the positive y-direction). Note that now the squares are no longer axis-aligned, but this will be no problem. Given a point $p \in \mathbb{R}^2$, we define the four *directional cones* $\mathcal{C}_\leftarrow(p)$,$\mathcal{C}_\uparrow(p)$, $\mathcal{C}_\rightarrow(p)$, and $\mathcal{C}_\downarrow(p)$ as the leftward, upward, rightward and downward cones with apex p and opening angle $\pi/2$. The directional cones subdivide the plane into four disjoint sectors. We will also need the *extended* rightward cone $\mathcal{C}^+_\rightarrow(p)$ with apex p and opening angle $\pi/2 + 16\varepsilon$.

CLAIM 4.7. *Let (u, v) be a directed pair in $\mathcal{P}_\phi$, and suppose that $\{p, q\}$ with $p \in P_u$ and $q \in P_v$ is the closest pair for (u, v). Then $\mathcal{C}_\uparrow(p) \cap P_u = \emptyset$ and $\mathcal{C}_\downarrow(q) \cap P_v = \emptyset$.*[11]

Proof. We prove the claim for $\mathcal{C}_\downarrow(q)$, the argument for $\mathcal{C}_\uparrow(p)$ is symmetric. We may assume that $|pq| = 1$.

[11] Recall that we set $\phi = \pi/2$, so $\uparrow$ and $\downarrow$ mean "in direction ϕ" and "in direction $-\phi$".

By assumption, the unit disk D through p contains no points of P_v, so it suffices to show that $\mathcal{C}_\downarrow(q) \cap S_v \subseteq D$. Since $\{u, v\} \in \mathcal{P}_\phi$ and by Observation 4.2, the direction of the line $\overline{pq}$ differs from ϕ by at most 17ε. Therefore, the intersections of the boundaries of $\mathcal{C}_\downarrow(q)$ and D have distance at least $\sqrt{2} - O(\varepsilon)$ from q. However, the pair $\{u, v\}$ is well-separated, so all points in P_v have distance at most ε from q, which implies the claim; see Figure 5. □

Given a set Z_u for a node u of T, we define the *upper chain* of Z_u, $\mathrm{UC}(Z_u)$ as follows: remove from Z_u all points p such that $\mathcal{C}_\uparrow(p)$ contains a point from Z_u in its interior. Then sort Z_u by x-coordinate and connect consecutive points by line segments. All segments of $\mathrm{UC}(Z_u)$ have slopes in $[-1, 1]$. Similarly, we define the *lower chain* of Z_u, $\mathrm{LC}(Z_u)$, by requiring the cones $\mathcal{C}_\downarrow(p)$ for the points in $\mathrm{LC}(Z_u)$ to be empty. The goal now is to compute $\mathrm{UC}(Z_u)$ and $\mathrm{LC}(Z_u)$ for all nodes u.

Define a directed graph Γ as follows: we create two copies of each vertex u in T, called $\texttt{start}(u)$ and $\texttt{end}(u)$, and we add a directed edge from $\texttt{start}(u)$ to $\texttt{end}(u)$ for each such vertex. Furthermore, we replace every edge uv of T (u being the parent of v) by two edges: one from $\texttt{start}(u)$ to $\texttt{start}(v)$, and one from $\texttt{end}(v)$ to $\texttt{end}(u)$. We call these edges the *tree*-edges. Finally, for every pair $\{u, v\} \in \texttt{wspd}(T)$, where S_v is wholly contained in the extended rightward cone $\mathcal{C}^+_\rightarrow(c_u)$, we create a directed edge from $\texttt{end}(u)$ to $\texttt{start}(v)$. These edges are called *wspd*-edges. Figure 4.1 shows a small example.

Claim 4.8. *The graph Γ is acyclic.*

Proof. Suppose C is a cycle in Γ. The tree-edges form an acyclic subgraph, so C has at least one wspd-edge. Let $e_1, e_2, \dots, e_z$ be the sequence of wspd-edges along C, and let $u_1, \dots, u_z$ be such that the endpoint of e_i is of the form $\texttt{start}(u_i)$. Finally, write $C = e_1 \to C_1 \to e_2 \to C_2 \to \cdots \to e_z \to C_z$, where C_i is the sequence of tree-edges between two consecutive wspd-edges. Each C_i consists of a (possibly empty) sequence of $\texttt{start} - \texttt{start}$ edges, followed by one $\texttt{start} - \texttt{end}$ edge and a (possibly empty) sequence of $\texttt{end}-\texttt{end}$ edges. Thus, the origin of the next wspd-edge e_{i+1} is an $\texttt{end}$-node for an ancestor or a descendant of u_i in T. In either case, by the definition of wspd-edges, it follows that the leftmost point of $S_{u_{i+1}}$ lies strictly to the right of the leftmost point of S_{u_i}. Thus, the leftmost point of $S_{u_{i+1}}$ lies strictly to the right of the leftmost point of S_{u_i} and the leftmost point of S_{u_1} lies strictly to the right of the leftmost point in S_{u_z}, which is absurd. □

Let $\leq_\Gamma$ be a topological ordering of the nodes of Γ.

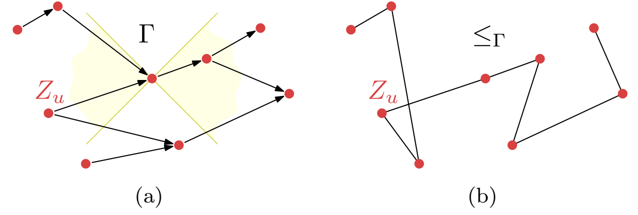

Figure 7: (a) A set of points, and all edges with a slope in $[-1, 1]$. By Claim 4.9, these edges are all (possibly implicitly) present in Γ. (b) A possible ordering $\leq_\Gamma$ of the points that respects Γ.

Claim 4.9. *Any pair (p, q) of points in Z_u with $p \leq_\Gamma q$ satisfies $q \notin \mathcal{C}_\leftarrow(p)$.*

Proof. Suppose for the sake of contradiction that $q \in \mathcal{C}_\leftarrow(p)$. Let v, w be the descendants of u such that $q \in P_v$, $p \in P_w$, and $\{v, w\} \in \texttt{wspd}(T)$. By Observation 4.2, S_w lies completely in the extended rightward cone $\mathcal{C}^+_\rightarrow(c_v)$, so Γ has an edge from $\texttt{end}(v)$ to $\texttt{start}(w)$. Now the tree edges in Γ require that the leaf with q comes before $\texttt{end}(v)$ and the leaf with p comes after $\texttt{start}(w)$, and the claim follows. □

Since all edges on $\mathrm{UC}(Z_u)$ have slopes in $[-1, 1]$, we immediately have the following corollary.

Corollary 4.10. *The ordering $\leq_\Gamma$ respects the orders of $\mathrm{UC}(Z_u)$ and $\mathrm{LC}(Z_u)$.* □

For every node $u \in T$, let $\leq_u$ be the order that $\leq_\Gamma$ induces on the leaf nodes corresponding to Z_u.

Claim 4.11. *All the orderings $\leq_u$ can be found in total time $O(n)$.*

Proof. To find the orderings $\leq_u$, perform a topological sort on Γ, in linear time[12] [23, Chapter 22.4]. With each node u of T store a list L_u, initially empty. We scan the nodes of Γ in order. Whenever we see a leaf for a point $p \in P$, we append p to the at most $2k$ lists L_u for the nodes u with $p \in Z_u$. The total running time is $O(n + \sum_{u \in T} |Z_u|) = O(n)$, and L_u is sorted according to $\leq_u$ for each $u \in T$. □

Claim 4.12. *For any node $u \in T$, if Z_u is sorted according to $\leq_u$, we can find $\mathrm{UC}(Z_u)$ and $\mathrm{LC}(Z_u)$ in time $O(|Z_u|)$.*

Proof. We can find $\mathrm{UC}(Z_u)$ by a Graham-type pass through L_u. An example of such a list is shown in

[12] Note that Γ has $O(n)$ edges, as $|\texttt{wspd}(T)| = O(n)$.

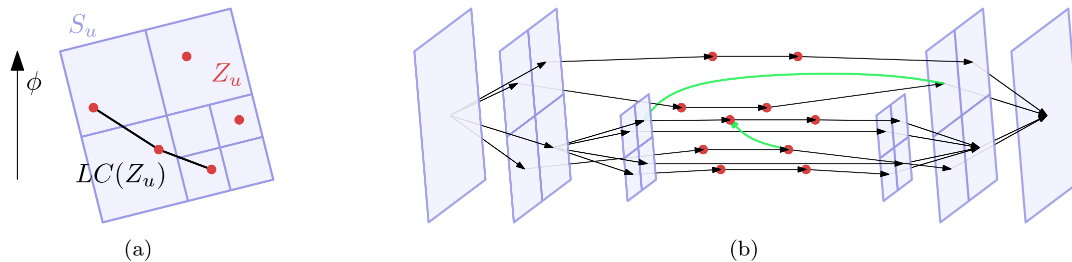

Figure 6: (a) A node u with $|Z_u| = 5$, and the relevant part of the quadtree. (b) The graph Γ. Tree edges are black (going right). To avoid clutter, we just show two wspd edges (green, going left).

Figure 7(b). That is, we scan L_u from left to right, maintaining a tentative upper chain U, stored as a stack. Let r be the rightmost point of U. On scanning a new point p, we distinguish cases depending in which of the four quadrants $\mathcal{C}_{\leftarrow}(r)$, $\mathcal{C}_{\uparrow}(r)$, $\mathcal{C}_{\rightarrow}(r)$, or $\mathcal{C}_{\downarrow}(r)$ it lies in. By Claim 4.1, we know that $p \notin \mathcal{C}_{\leftarrow}(r)$. If $p \in \mathcal{C}_{\downarrow}(r)$, we discard p and continue to the next point in L_u. If $p \in \mathcal{C}_{\uparrow}(r)$, we pop r from U and reassess p from the point of view of the new rightmost point of U. If $p \in \mathcal{C}_{\rightarrow}(r)$, we push p onto U.

The algorithm takes $O(|Z_u|)$ time, because every point is pushed or popped from the stack at most once and because it takes constant time to decide which point to push or pop. For correctness, it suffices to observe that the conflict set of any new point p along the current upper chain must be contiguous. The lower chain is computed in an analogous manner. □

Claim 4.13. *For any node $u \in T$ and any pair $\{u, v\}$ in $\mathcal{P}_u$, given $\mathrm{UC}(Z_u)$ and $\mathrm{LC}(Z_v)$, we can find the closest pair in $Z_u \otimes Z_v$ in time $O(|Z_u| + |Z_v|)$.*

Proof. Connect the endpoints of $\mathrm{UC}(Z_u)$ and $\mathrm{LC}(Z_v)$ to obtain a simple polygon (note that the two new edges cannot intersect the chains, because $\{u, v\}$ has direction $\phi = \pi/2$, so by Observation 4.2 $\Phi_{uv} \subseteq [\pi/2 - 8\frac{1}{2}\varepsilon, \pi/2 + 8\frac{1}{2}\varepsilon]$ and all edges of the chains have slopes in $[-1, 1]$). Then use the algorithm of Chin and Wang [21] to find the constrained DT of the polygon in time $O(|Z_u| + |Z_v|)$. The closest pair will appear as an edge in this DT, and hence can be found in the claimed time.[13] □

Lemma 4.14. *In total linear time, we can find for every $u \in T$ and for every pair $\{u, v\} \in \mathcal{P}_u$ the closest pair in $Z_u \otimes Z_v$.*

[13] Actually, the resulting polygon is x-monotone, so the most difficult part of the algorithm by Chin and Wang [21], finding the visibility map of the polygon [17], becomes much easier [32]. The problem may allow a much more direct solution, but since we will later require Chin and Wang's algorithm in full generality, we omit these details.

Proof. By Claims 4.11, 4.12, 4.13, the time to find all the closest pairs is $O(n + \sum_{u\in T} \sum_{\{u,v\}\in\mathcal{P}_u} (|Z_u| + |Z_v|)) = O(n + \sum_{u\in T} |Z_u|) = O(n)$, because every v appears in only constantly many $\mathcal{P}_u$'s. □

We thus obtain the main result of this section.

Theorem 4.15. *Given a compressed quadtree T for P and $\mathtt{wspd}(T)$, we can find a graph H with $O(n)$ edges such that H contains all edges of $\mathrm{emst}(P)$. It takes $O(n)$ time to construct H.*

Proof. The fact that H contains the EMST follows from Lemmas 4.4, 4.5 and 4.6. The running time follows from the discussion at the beginning of Steps 1 and 2 and Lemma 4.14. □

4.2 Extracting the EMST. At this point, we have a sparse graph H on P that contains all EMST edges. Furthermore, we can use Theorem 2.1 to convert our compressed quadtree into a c-cluster quadtree T, such that each edge is associated with two squares of T. (This information can be preserved during the transformation from compressed quadtree to c-cluster quadtree, because every old square overlaps with only constantly many new squares of similar size.) We can ensure that every edge e of H is associated with squares S such that the length of e is between $|S|/2\varepsilon$ and $2|S|/\varepsilon$, by using Claim 2.5 and inserting additional nodes into T if necessary. Note that these nodes can be inserted in the right order in total linear time, because by construction every node of T is associated only with constantly many edges. We want to extract $\mathrm{emst}(P)$, but in general no deterministic linear time algorithm for this problem is known: the fastest deterministic algorithm whose running time can be analyzed needs $O(n\alpha(n))$ steps [18]. However, the special structure of H and the c-cluster quadtree T make it possible to achieve linear time.

By the cut property of minimum spanning trees, $\mathrm{emst}(P)$ is connected within each c-cluster. Thus, we can process the clusters bottom-up, and we only need to

find the EMST within a c-cluster given that the points in each child are already connected. Within this cluster, T is a regular uncompressed quadtree, and we can use the structure of T to perform an appropriate variant of Borůvka's MST algorithm [8, 50] in linear time.

LEMMA 4.16. *Let T' be a subtree of T corresponding to a c-cluster, and let E be the edges in H associated with T'. Then $\text{emst}(P) \cap E$ can be computed in time $O(|E| + |V(T')|)$.*

Proof. Let ℓ be the size of the root square of T'. Through a level order traversal of T' we group the squares in $V(T')$ by height into layers $V_1, V_2, \ldots, V_h$ (where V_1 is the bottommost layer, and V_h contains only the root). The squares in V_i have size $\ell/2^{h-i}$. Every square has a set of associated edges in E. By Claim 2.5 (and the fact that T' is a regular quadtree so that the parent of S has size at most $2|S|$), the edges assigned to a square S have length between $|S|/2\varepsilon$ and $2|S|/\varepsilon + 2|S| \leq 3|S|/\varepsilon$, for ε small enough. To find the EMST, we subdivide the edges into sets E_i, where E_i contains all edges with length in $[\ell/(\varepsilon 2^{h-i}), \ell/(\varepsilon 2^{h-i-1}))$. Given the V_i, we can determine the sets E_i in total time $O(|E| + |V(T')|)$, as the edges for E_i are associated only with squares in $V_{i-\alpha}, V_{i-\alpha+1}, \ldots, V_{i+\alpha}$, for some constant α. For each edge $e \in E_i$, we find the squares in V_i that contain the endpoints of e. This takes constant time, as we need to inspect only constantly many levels above or below the squares associated with e. Thus, the total work is $O(|E| + |V(T')|)$. Note that every edge in E_i is crossed by $O(1)$ other edges in E_i, because all $e \in E_i$ have roughly the same length and because every pair of squares in V_i has only constantly many associated edges in E_i.

Now we compute the EMST by processing the sets $E_1, \ldots, E_h$ in order. Here is how to process E_i. We consider the squares in V_i. Assume that we know for each square of V_i the connected component in the current partial EMST it meets (initially each c-cluster is its own component). By the cut property, every square S meets only one connected component, as S is much smaller than the edges in E_i. Eliminate all edges in E_i between squares in the same component, and remove duplicate edges between each two components, keeping only the shortest of these edges (this takes $O(|E_i|)$ time with appropriate pointer manipulation). Then find the shortest edge out of each component and add these edges to the partial EMST. Determine the new components and merge their associated edge sets. This sequence of steps is called a *Borůvka-phase*. Perform Borůvka-phases until E_i has no edges left.

By the crossing-number inequality, the number of edges considered in each phase is proportional to the number r of components with an outgoing edge in that phase. Indeed, viewing each component as a supervertex, we have an embedding of a graph with r vertices and z edges such that there are $O(z)$ crossings (since every edge $e \in E_i$ is crossed by $O(1)$ other edges in E_i). Thus, the crossing number inequality [43] yields $z^3/r^2 \leq \beta z$, for some constant $\beta > 0$, so $z = O(r)$. Since the number of components at least halves in each phase, and since initially there are at most $|V_i|$ components, the total time for E_i is $O(|E_i| + |V_i|)$. Finally, label each square in V_{i+1} with the component it meets and proceed with round $i+1$. In total, processing T takes time $O(|V(T')| + |E|)$, as desired. □

We conclude:

THEOREM 4.17. *Let P be a planar point set and T be a compressed quadtree or a c-cluster quadtree for P. Then $\text{DT}(P)$ can be computed in time $O(|P|)$.*

Proof. If T is a c-cluster quadtree, invoke Theorem 2.1 to convert it to a compressed quadtree. Then use Theorem 2.2 to obtain $\texttt{wspd}(T)$. Next, apply Theorem 4.15 to compute the supergraph H of $\text{emst}(P)$. After that, if necessary, convert T to a c-cluster quadtree for P via Theorem 2.1, and apply Lemma 4.16 to each c-cluster, in a bottom-up manner, to extract $\text{emst}(P)$. Finally, apply the algorithm by Chin and Wang [21] to find $\text{DT}(P)$. All this takes time $O(|P|)$, as claimed . □

5 Applications

Our result yields deterministic versions of several recent randomized algorithms related to DTs. Firstly, we can immediately derandomize an algorithm for hereditary DTs by Chazelle *et al.* [19, 20]:

COROLLARY 5.1. *Let P a planar n-point set, and let $S \subseteq P$. Given $\text{DT}(P)$, we can find $\text{DT}(S)$ in deterministic time $O(n)$ on a pointer machine.*

Proof. Use Theorem 3.1 to find a c-cluster quadtree T for P, remove the leaves for $P \setminus S$ from T and trim it appropriately.[14] Finally, apply Theorem 4.17 to extract $\text{DT}(S)$ from T, in time $O(n)$. □

Secondly, we obtain deterministic analogues of the algorithms by Buchin *et al.* [9] to preprocess imprecise point sets for faster DTs. For example, we can prove the following:

[14] Deleting $P \setminus S$ might create new c-clusters. However, since we are aiming for running time $O(n)$, we can apply Theorem 4.17 to a partly compressed quadtree that may contain long paths where every node has only one child.

Corollary 5.2. *Let $\mathcal{R} = \langle R_1, R_2, \ldots, R_n \rangle$ be a sequence of n β-fat planar regions so that no point in $\mathbb{R}^2$ meets more than k of them. We can preprocess $\mathcal{R}$ in $O(n \log n)$ deterministic time into an $O(n)$-size data structure so that given a sequence of n points $P = \langle p_1, p_2, \ldots, p_n \rangle$ with $p_i \in R_i$ for all i, we can find $\mathrm{DT}(P)$ in deterministic time $O(n \log(k/\beta))$ on a pointer machine.*

Proof. The method of Buchin *et al.* [9, Theorem 4.3 and Corollary 5.6] proceeds by computing a representative quadtree T for $\mathcal{R}$. Given P, the algorithm finds for every point in P the leaf square of T that contains it, and then uses this information to obtain a compressed quadtree T' for P in time $O(n \log(k/\beta))$. However, T' is *skewed* in the sense that not all its squares need to be perfectly aligned and that some squares can be cut off. However, the authors argue that even in this case $\mathtt{wspd}(T)$ takes $O(n)$ time and yields a linear-size WSPD [9, Appendix B]. The main observation [9, Observation B.1] is that any (truncated) square S in T' is adjacent to at least one square whose area is at least a constant fraction of the area S would have without clipping. Since in skewed quadtrees the size of a node is at most half the size of its parent, the argument of Lemma 4.4 still applies. To see that Lemma 4.5 still holds, we need to check that the volume argument still goes through. For this, note that by the main observation of Buchin *et al.*, we can assign every square R_w (the notation is as in the proof of Lemma 4.5) to an adjacent square of comparable size at distance $O(\varepsilon)$ from A. Since every such square is charged by disjoint descendants from constantly many neighbors, the volume argument still applies, and Lemma 4.5 still holds. Lemma 4.14 only relies on well-separation and the combinatorial structure of T, and hence is still valid. Finally, in order to apply Lemma 4.16, we need to turn T' into a c-cluster quadtree, which takes linear time by Theorem 2.1. Thus, the total running time is $O(n \log(k/\beta)$, as claimed. □

Finally, Buchin and Mulzer [10] showed that for word RAMs, DTs are no harder than sorting. We can now do it deterministically. Let $\mathtt{sort}(n)$ be the time to sort n integers on a w-bit word RAM. The best deterministic bound for $\mathtt{sort}(n)$ is $O(n \log \log n)$ [33].[15]

Corollary 5.3. *Let P be a planar n-point set given by w-bit integers, for some $w \geq \log n$. We can find $\mathrm{DT}(P)$ in deterministic time $O(\mathtt{sort}(n))$ on a word RAM supporting the* `shuffle`*-operation.*[16]

Proof. Buchin and Mulzer [10] show how to find a compressed quadtree T for P in time $O(\mathtt{sort}(n))$, using the `shuffle`-operation. They actually do not find the squares of the quadtree, only the combinatorial structure of T and the bounding boxes B_v. It is easily seen that the algorithm `wspd` also works in this case.

To apply Lemma 4.4, we need to check that the sizes of the bounding boxes decrease geometrically down the tree. For this, consider a node $v \in T$ with associated point set P_v and the quadtree square S_v (i.e., the smallest aligned square of size 2^l such that the coordinates of all points in P_v share the first $w - l$ bits). Let B_v be the bounding box of P_v, and let l' be such that $2^{l'+1} \geq |B_v| \geq 2^{l'}$. Clearly, B_v meets at most nine aligned squares of size $2^{l'}$, arranged in a 3×3 grid. Hence, any descendant $\tilde{v}$ of v that is at least five levels below v must have $|B_{\tilde{v}}| \leq |S_{\tilde{v}}| \leq |B_v|/2$, since after at most four (compressed) quadtree divisions the squares for B_v have been separated. Thus, the proof of Lemma 4.4 goes through as before, if we choose k larger and consider every fifth node along the chain $u_1, u_2, \ldots, u_k, u$.

Lemma 4.5 still holds, because every bounding box B_v is contained in a (possibly much larger) square S_v, so the volume argument still applies. Furthermore, Lemma 4.14 only relies on well-separatedness and the combinatorial structure of T, so we can find the graph H in linear time. After that, it takes $O(n)$ time to compute $\mathrm{emst}(P)$, using the transdichotomous minimum spanning tree algorithm by Fredman and Willard [30]. □

Acknowledgments

This work was initiated while the authors were visiting the Computational Geometry Lab at Carleton University. We would like to thank the group at Carleton and especially our hosts Jit Bose and Pat Morin for their wonderful hospitality and for creating a great research environment. We also would like to thank Kevin Buchin and Martin Nöllenberg for sharing their thoughts on this problem with us. Work on this paper by M. Löffler has been supported by the Office of Naval Research under MURI grant N00014-08-1-1015. Work by W. Mulzer has been supported in part by NSF grant CCF-0634958, NSF CCF 083279, and a Wallace Memorial Fellowship in Engineering.

[15] For specific ranges of w, we can do better. For example, if $w = O(\log n)$, radix sort shows that $\mathtt{sort}(n) = O(n)$ [23].

[16] For two w-bit words, $x = x_1 \ldots x_w$ and $y = y_1 \ldots, y_w$, we define $\mathtt{shuffle}(x, y)$ as the $2w$-bit word $z = x_1 y_1 x_2 y_2 \ldots x_w y_w$.

References

[1] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(5):407–422, 1991.

[2] N. Ailon, B. Chazelle, K. L. Clarkson, D. Liu, W. Mulzer, and C. Seshadhri. Self-improving algorithms. `arXiv:0907.0884`, 2009.

[3] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Self-improving algorithms. In *Proc. 17th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 261–270, 2006.

[4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag, Berlin, third edition, 2008.

[5] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *J. Comput. System Sci.*, 48(3):384–409, 1994.

[6] M. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. *Internat. J. Comput. Geom. Appl.*, 9(6):517–532, 1999.

[7] J.-D. Boissonnat and M. Yvinec. *Algorithmic geometry*. Cambridge University Press, Cambridge, 1998.

[8] O. Borůvka. O jistém problému minimálním. *Práce Moravské Přírodovědecké Společnosti*, 3:37–58, 1926.

[9] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Preprocessing imprecise points for Delaunay triangulation: Simplified and extended. *Algorithmica*. To appear. Preliminary version in WADS'09.

[10] K. Buchin and W. Mulzer. Delaunay triangulations in $O(\texttt{sort}(n))$ time and more. *J. ACM*. To appear. Preliminary version in FOCS'09.

[11] A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, and J. R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM J. Comput.*, 38(4):1533–1573, 2008.

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

[13] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, 1995.

[14] T. M. Chan. Well-separated pair decomposition in linear time? *Inform. Process. Lett.*, 107(5):138–141, 2008.

[15] T. M. Chan and M. Pătraşcu. Voronoi diagrams in $n \cdot 2^{O(\sqrt{\lg\lg n})}$ time. In *Proc. 39th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 31–39, 2007.

[16] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry. I. Point location in sublogarithmic time. *SIAM J. Comput.*, 39(2):703–729, 2009.

[17] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[18] B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *J. ACM*, 47(6):1028–1047, 2000.

[19] B. Chazelle, O. Devillers, F. Hurtado, M. Mora, V. Sacristán, and M. Teillaud. Splitting a Delaunay triangulation in linear time. *Algorithmica*, 34(1):39–46, 2002.

[20] B. Chazelle and W. Mulzer. Computing hereditary convex structures. In *Proc. 25th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 61–70, 2009.

[21] F. Chin and C. A. Wang. Finding the constrained Delaunay triangulation and constrained Voronoi diagram of a simple polygon in linear time. *SIAM J. Comput.*, 28(2):471–486, 1999.

[22] K. L. Clarkson and C. Seshadhri. Self-improving algorithms for Delaunay triangulations. In *Proc. 24th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 148–155, 2008.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009.

[24] G. Das and D. Joseph. Which triangulations approximate the complete graph? In *Proceedings of the international symposium on Optimal algorithms*, pages 168–192, 1989.

[25] B. Delaunay. Sur la sphère vide. A la memoire de Georges Voronoi. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, 7:793–800, 1934.

[26] O. Devillers. Delaunay triangulation of imprecise points, preprocess and actually get a fast query time. Technical Report 7299, INRIA, 2010. `http:// hal.archives-ouvertes.fr/ docs/ 00/ 48/ 59/ 15/PDF/RR-7299.pdf`.

[27] D. Eppstein. Spanning trees and spanners. In *Handbook of computational geometry*, pages 425–461. North-Holland, Amsterdam, 2000.

[28] D. Eppstein, M. Goodrich, and J. Sun. The skip quadtree: a simple dynamic data structure for multidimensional data. *Internat. J. Comput. Geom. Appl.*, 18(1–2):131–160, 2008.

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

[30] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. System Sci.*, 48(3):533–551, 1994.

[31] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Systems Theory*, 17(1):13–27, 1984.

[32] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Inform. Process. Lett.*, 7(4):175–179, 1978.

[33] Y. Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, 2004.

[34] S. Har-Peled. Quadtrees — hierarchical grids. Lecture notes, 2010.

[35] S. Har-Peled. Well separated pair decompositions. Lecture notes, 2010.

[36] D. Harel and R. E. Tarjan. Fast algorithms for

finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[37] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete Comput. Geom.*, 7(1):13–28, 1992.

[38] D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, 3rd edition, 1997.

[39] D. Krznaric and C. Levcopoulos. Computing hierarchies of clusters from the Euclidean minimum spanning tree in linear time. In *Proc. 15th Found. Software Technology and Theoret. Comput. Sci. (FSTTCS)*, pages 443–455, 1995.

[40] D. Krznaric and C. Levcopoulos. Computing a threaded quadtree from the Delaunay triangulation in linear time. *Nordic J. Comput.*, 5(1):1–18, 1998.

[41] D. Krznaric, C. Levcopoulos, and B. J. Nilsson. Minimum spanning trees in d dimensions. *Nordic J. Comput.*, 6(4):446–461, 1999.

[42] M. Löffler and J. Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Comput. Geom. Theory Appl.*, 43(3):234–242, 2010.

[43] J. Matoušek. *Lectures on discrete geometry.* Springer-Verlag, New York, 2002.

[44] O. Musin. Properties of the Delaunay triangulation. In *Proc. 13th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 424–426, 1997.

[45] F. P. Preparata and M. I. Shamos. *Computational geometry. An Introduction.* Springer-Verlag, New York, 1985.

[46] E. Pyrga and S. Ray. New existence proofs for ϵ-nets. In *Proc. 24th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 199–207, 2008.

[47] H. Samet. *The design and analysis of spatial data structures.* Addison-Wesley, Boston, MA, USA, 1990.

[48] A. Schönhage. On the power of random access machines. In *Proc. 6th Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 520–529, 1979.

[49] M. I. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 151–162, 1975.

[50] R. E. Tarjan. *Data structures and network algorithms.* SIAM, Philadelphia, 1983.

[51] K. R. Varadarajan. A divide-and-conquer algorithm for min-cost perfect matching in the plane. In *Proc. 39th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 320–331, 1998.

[52] A. C. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

A Computational Models

Since our results concern different computational models, we use this appendix to describe them in more detail. Our two models are the real RAM/pointer machine and the word RAM.

The Real RAM/Pointer Machine. The standard model in computational geometry is the *real RAM*. Here, data is represented as an infinite sequence of storage cells. These cells can be of two different types: they can store real numbers or integers. The model supports standard operations on these numbers in constant time, including addition, multiplication, and elementary functions like square-root. The *floor* function can be used to truncate a real number to an integer, but if we were allowed to use it arbitrarily, the real RAM could solve PSPACE-complete problems in polynomial time [48]. Therefore, we usually have only a restricted floor function at our disposal, and in this paper it will be banned altogether.

The *pointer machine* [38] models the list processing capabilities of a computer and disallows the use of constant time table lookup. The data structure is modeled as a directed graph G with bounded outdegree. Each node in G represents a *record*, with a bounded number of pointers to other records and a bounded number of (real or integer) data items. The algorithm can access data only by following pointers from the inputs (and a bounded number of global entry records); random access is not possible. The data can be manipulated through the usual real RAM operations (again, we disallow the floor function).

Word RAM. The *word RAM* is essentially a real RAM without support for real numbers. However, on a real RAM, the integers are usually treated as atomic, whereas the word RAM allows for powerful bit-manipulation tricks. More precisely, the word RAM represents the data as a sequence of w-bit words, where $w \geq \log n$ (n being the problem size). Data can be accessed arbitrarily, and standard operations, such as Boolean operations (`and`, `xor`, `shl`, $\ldots$), addition, or multiplication take constant time. There are many variants of the word RAM, depending on precisely which instructions are supported in constant time. The general consensus seems to be that any function in AC^0 is acceptable.[17] However, it is always preferable to rely on a set of operations as small, and as non-exotic, as possible. Note that multiplication is not in AC^0 [31]. Nevertheless, it is usually included in the word RAM instruction set [30].

[17] AC^0 is the class of all functions $f : \{0,1\}^* \to \{0,1\}^*$ that can be computed by a family of circuits $(C_n)_{n\in\mathbb{N}}$ with the following properties: (i) each C_n has n inputs; (ii) there exist constants a, b, such that C_n has at most an^b gates, for $n \in \mathbb{N}$; (iii) there is a constant d such that for all n the length of the longest path from an input to an output in C_n is at most d (i.e., the circuit family has bounded depth); (iv) each gate has an arbitrary number of incoming edges (i.e., the *fan-in* is unbounded).

B On Quadtrees

B.1 c-Cluster Quadtrees. We elaborate on a few more properties of the c-cluster trees and c-cluster quadtrees defined in Section 2.2. Krznaric and Levcopolous [39, Theorem 7] showed that a c-cluster tree can be computed in linear time from a Delaunay triangulation.

THEOREM B.1. (KRZNARIC-LEVCOPOLOUS) *Let P be a planar n-point set. Given a constant $c \geq 1$ and $\mathrm{DT}(P)$, we can find a c-cluster tree T_c for P in $O(n)$ time and space on a pointer machine.* □

The c-cluster tree T_c is quite similar to a well-separated pair decomposition: all of its unrelated nodes are mutually $(1/c)$-well-separated. However, T_c can be much too weak, since nodes in T_c may have arbitrarily many children. For example, if the points in P lie on a grid, then T_c consists of a single root with n children. Nonetheless, T_c gives us valuable information, since it represents a decomposition of the point set into pieces of bounded spread. Therefore, we define the c-cluster quadtree T, which is obtained by augmenting the c-cluster tree with quadtree-like pieces that replace each high-degree node in T_c. For a node u of T_c, let T_u^Q be the balanced regular quadtree on the representative points of u's children in T_c. Let v be one of those children. Since the children of u are mutually well-separated, we can show:

OBSERVATION B.2. *If c is sufficiently large, at most four leaf squares of T_u^Q can contain points from P_v.*

Proof. Let $d = |B_v|$ be the diameter of the bounding box of the points in P_v. By construction, P_v is c-semi-separated from the rest of P, that is, the distance from any point in P_v to any other point is at least cd. Now, let $p \in P_v$ be the chosen representative point. Any leaf node ν of T_u^Q that can potentially contain points of P_v must be within distance d of p. Furthermore, any such square must have diameter at least $|S_\nu| \geq \frac{1}{2}(c-1)d$, since otherwise its parent could not possibly contain any points other than p, and therefore would not have been subdivided. Now, clearly, there cannot be more than four such squares if they are all larger than $2d$, that is, the observation is true whenever $\frac{1}{2}(c-1)d \geq 2d$, i.e. when $c \geq 5$. □

To see that c-cluster quadtrees have linear size, we need a property that is (somewhat implicitly) shown in [40, Section 4.3].

LEMMA B.3. *If u has m children $v_1, v_2, \ldots, v_m$ in T_c, then T_u^Q has $O(m)$ nodes.*

Proof. Call a square $S \in T_u^Q$ *full* if $S \cap P_u \neq \emptyset$. Clearly, it suffices to bound the number of full squares. Call a full square $S \in T_u^Q$ *merged* if it has at least two full children. Since there are $O(m)$ merged squares, we only need to bound the number of non-merged squares. For this we use the following claim to charge non-merged squares to merged squares. The *direct neighbors* of a square S in T_u^Q are the 8 squares of size $|S|$ adjacent to S.

CLAIM B.4. *Let S be a full square. There exists a constant β (depending on c), such that at least one of the β closest ancestors of S (possibly S itself) is either merged or has a merged direct neighbor.*

Proof. If S contains only a single point, then its parent must be a merged node, otherwise S would not be part of the tree. Otherwise, S contains points from more than one P_{v_i}, so $S \cap P_u$ is not a c-cluster, and there has to be a point from $P_u \setminus S$ at distance at most $c|S|$ from S. Hence, after constantly many levels up the tree, we arrive at an ancestor S' of S with a full direct neighbor $S'' \neq S'$. If S' and S'' are merged at the next step, we are done, so we assume this does not happen. In this case, since $(S' \cup S'') \cap P_u$ is not a c-cluster, there is a point in $P_u \setminus (S' \cup S'')$ at distance at most $cD(S' \cup S'') \leq 2c|S'|$, contained in a square of size at most $(c/4)|S'|$. Thus, again after constantly many levels up the tree, if no merge occurs, we find a connected set of three full squares (connected with respect to the direct neighborhood adjacency relationship). This can repeat at most twice more, because any connected set of at least five full squares at the same level of a quadtree must have a merged node among its ancestors. □

Now, we charge each full non-merged node to the merged node as in the claim. Since each merged node can be charged at most $9 \cdot 4^\beta$ times this way, the Lemma B.3 follows. □

B.2 Equivalence of Compressed and c-Cluster Quadtrees. Before proving Theorem 2.1, we will recall some known facts and make some observations. First, we make an observation about (uncompressed) quadtrees, which says we can realign a quadtree locally any way we want. We call a quadtree for P *λ-relaxed* if it has at most λ points of P in each leaf, and is otherwise a normal quadtree.

LEMMA B.5. *Let P a set of points and T a regular quadtree for P, with base square R. Let S be another square with $P \subseteq S$ and $|S| = O(|R|)$. Then we can build a 4-relaxed quadtree T' for P with base square S in $O(|T|)$ time such that T' has $O(|T|)$ nodes.*

Proof. First, if S is larger than R, we subdivide S in quadtree-like fashion until we get a grid with squares of size between $|R|/2$ and $|R|$. If S is smaller than R, we keep doubling its size, keeping the top left corner fixed, again until it is of size between $|R|/2$ and R. By assumption, this takes constant time. Then we determine the squares $S'_1, \ldots, S'_k$ of the current grid that intersect R (note that $k \leq 9$). Let $L := \{S'_1, \ldots, S'_k\}$. The set L contains all the *active* squares of T'. We will maintain the invariant that each active square S' is associated with a set $\mathrm{as}(S')$ of constantly many squares of T that intersect it. Initially, we set $\mathrm{as}(S'_1) = \cdots = \mathrm{as}(S'_k) = \{R\}$. Now the algorithm proceeds as follows: remove a square S' from L. Subdivide S' into four squares S'_1, S'_2, S'_3, and S'_4. Let C be the set of children of the nodes in $\mathrm{as}(S')$. For each S'_i, determine which squares in C intersect S'_i, if any. This can be done in constant time, because $|\mathrm{as}(S')| = O(1)$. Now, we associate S'_i with the squares in C that intersect it, and also with the squares in $\mathrm{as}(S')$ that have no children and intersect S'_i. If S'_i intersects at least one square in C, we add it to L, otherwise we declare it a *relaxed leaf*. This process continues until L becomes empty.

Clearly, for any square S' in L we have that $\mathrm{as}(S')$ contains only squares that intersect S' and have size at least $|S'|$. Furthermore, at least one square in $\mathrm{as}(S')$ has size at most $4|S'|$ (we call such a square a *principal* square for S'). Thus, we maintain the invariant that each square in L has $O(1)$ associated squares. Conversely, since any square of T can be a principal square for only $O(1)$ many squares that appear in L, the total number of squares that appear in L, and hence the total time for the refinement process, is proportional to the number of nodes in T.

Finally, we process the relaxed leaves of T': each relaxed leaf S' is associated with $O(1)$ leaves of T. Thus, we can find in constant time the $O(1)$ points of P that are contained in S'. □

A quadtree is said to be *balanced* if for every node u that is either a leaf or a compressed node, the square S_u is only adjacent to squares that are within a factor 2 of the size of S_u. It is well known that regular (uncompressed) quadtrees can be balanced in linear time:

THEOREM B.6. (THEOREM 14.4 OF [4]) *Let T be a quadtree with m nodes. Then the balanced version of T has $O(m)$ nodes and can be constructed in $O(m)$ time.* □

Furthermore, we can also add pointers between the (constantly many) adjacent squares of each square during the construction. It is not hard, but cumbersome, to extend the result to compressed quadtrees.

LEMMA B.7. *Let T be a compressed quadtree with m nodes. Then the balanced version of T has $O(m)$ nodes and can be constructed in $O(m)$ time.*

Proof. We provide an algorithm for a more general problem: let T be a compressed quadtree for P with m nodes and base square R. Let S be another square with $P \subseteq S$ and $|S| = O(|R|)$. The goal is to build a balanced compressed quadtree T' for P with base square S. For this we proceed similarly as in the proof of Lemma B.5: first we subdivide S until we get a grid with squares of size between $|R|/2$ and $|R|$. As before, we then maintain a list L of active squares, initialized with the squares of the current grid that intersect R, and we maintain for each active square S' in T' a list $\mathrm{as}(S')$ of constantly many associated squares in T. Now $\mathrm{as}(S')$ may also contain compressed children that meet S'.

When processing an active square S', we first proceed as in the proof of Lemma B.5: subdivide S' into four children and consider the children of the regular nodes in $\mathrm{as}(S')$, updating the associations as before. We then consider the compressed children in $\mathrm{as}(S')$ and associate them with the children of S' that meet them. Finally, if $\mathrm{as}(S')$ contains compressed nodes, we consider their compressed children and associate them with the appropriate children of S'. A square S' becomes inactive once it has no associated nodes of size between $|S'|$ and $2|S'|$ and no associated compressed child of size as least $|S|/2^{5a}$. After that, we check the four neighbors of S' (directly to the left, right, above, and below). If any of these neighbors N is not active and has size at least $2|S'|$, we subdivide N into four children, and update the information about which children contain the points in $N \cap P$. Since N is not active, we have $|N \cap P| = O(1)$, so this takes constant time. We keep the new squares inactive, unless N meets a compressed child that is larger than $|N|/2^{5a}$. In this case, we make the children of N that meet the compressed child active and insert them into L appropriately (e.g., right after S'). If necessary, we propagate this step to the neighbors of N to ensure that the quadtree is balanced.

We continue this process until L is empty. At that point all the relaxed leaves of T' meet constantly many points in P or compressed children of T (at most four). We then construct a local compressed quadtree for each relaxed leaf, creating new compressed nodes if necessary. Finally, we recurse on the remaining compressed children. Each such child can be associated with at most four relaxed leaves, and by choosing the new bounding box in the recursion appropriately, we can align them with the parent tree.

All this takes time $O(m)$, because each split can be charged to an active square in L, and each active square has either an associated principal square or an

associated large cluster child in T, and by construction each node of T can be associated with $O(1)$ active squares. □

We now present the proof of Theorem 2.1 in two lemmas.

LEMMA B.8. *Let P be a set of points in the plane. Given a c-cluster quadtree on P, we can compute in linear time an $O(c)$-compressed quadtree on P.*

Proof. We will build the compressed quadtree top-down, starting with the root of the c-cluster tree. Consider a node u with children V in the c-cluster tree, and let q be the representative point of P_u in the quadtree of u's parent, and Q the set of representative points of the nodes in V for u. Suppose we have already built a valid compressed quadtree for a set of points that includes q. We will show how to change it into a compressed quadtree where the other points of Q have also been added.

Let T be the current quadtree, and T_u^Q the quadtree in the c-cluster quadtree associated with u (which has Q as its points). We know that the root box of T_u^Q must be much smaller than the leaf of T that contains q, since P_u is $(1/c)$-semi-separated from the rest of P. So, if the root box of T_u^Q falls completely within this leaf square, we just add it as a child. However, by Observation B.2 it is also possible that it intersects up to four leaf squares of T. If this is the case, we identify a square at most twice the size of T_u^Q's root box that is aligned appropriately with the relevant edges of T, and apply Lemma B.5 to realign T_u^Q with T. Similarly, if T_u^Q's root box falls partially outside the rootbox of T, we simply grow T to become four times as big and then do the same thing.

Note that once we add a node to T, it stays there, and we only realign the small local quadtrees. This takes linear time in their sizes, so the total time spent is linear. □

LEMMA B.9. *Let P be a set of points in the plane. Given an a-compressed quadtree on P, we can compute in linear time a $O(a)$-cluster quadtree on P.*

Proof. We are given a compressed quadtree on P, and have to determine the c-clusters and reuse the quadtree pieces for the high-degree nodes of the c-cluster tree. First we balance the tree according to Lemma B.7. If a set $C \subset P$ is a c-cluster for, say, $c = 2a$, then the nodes in the compressed quadtree containing C must be compressed nodes, and there can be at most four such nodes. The converse is not true: the compressed quadtree may contain compressed nodes that are not part of c-clusters. Nonetheless, we can identify the clusters by traversing the tree top-down and for each node with a compressed child check in constant time whether any of its neighbors also have compressed children, since the tree is balanced, and whether any combination of these child point sets forms a c-cluster. We proceed down the tree until all c-clusters have been found.

Now, for each node u in the c-cluster tree, we reuse part of the compressed quadtree to create a quadtree of the representative points of u's children. Since P_u is divided into at most four pieces in the compressed quadtree, we can take those four compressed nodes and realign them according to Lemma B.5 and add a new root above them. The new relaxed tree can have constantly many points in each leaf. We now first select an arbitrary representative point for each of the point sets P_v for $v \in V$ a child of u in the c-cluster tree, and prune the quadtree of any leaves that do not contain such a representative point. This takes time $O(|V|)$ for one node, so linear time in total. Finally, we refine the leaves that still contain multiple points using a standard quadtree. By Lemma B.3 we know that the total tree has linear size, so this takes linear time. □

C Adapting the Algorithm by Krznaric and Levcopolous to the Pointer Machine

We now describe how to adapt the algorithm by Krznaric and Levcopolous (KL) for finding a c-cluster quadtree for a planar point set P given $\mathrm{DT}(P)$. The goal is to make the algorithm work on a pointer machine/real RAM.

C.1 Terminology. We begin by recalling some terminology from KL.

- **neighborhood.** The *neighborhood* of a square S of a quadtree consists of the 25 squares of size $|S|$ concentric around S (including S); see Figure 8.
- **direct neighborhood.** The *direct neighborhood* of a square S consists of the 9 squares of size $|S|$ directly adjacent to S (including S); see Figure 8.
- **star of a square.** Let P be a planar point set, and let S be a square. The *star* of S, denoted by $\bigstar(S)$, is the set of all edges e in $\mathrm{DT}(P)$ such that (i) e has one endpoint inside S and one endpoint outside the neighborhood of S; and (ii) $|e| \leq 16|S|$.
- **dilation.** Let P be a planar point set, and G a connected plane graph with vertex set P. The *dilation* of P is the distortion between the shortest path metric in G and the Euclidean distance, i.e.,

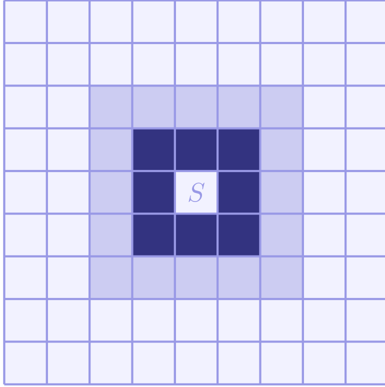

Figure 8: The neighborhood of a square S. The direct neighbors are shown in dark blue, the others in light blue.

the maximum ratio, over all pairs of distinct points $p, q \in P$, between the length of the shortest path in G from p to q, and $|pq|$. There are many families of planar graphs whose dilation is bounded by a constant [24]. In particular, the dilation for any planar point set P, the dilation of $\mathrm{DT}(P)$ is bounded by $2\pi/(3\cos(\pi/6)) \leq 2.42$ [37].

- **orientation.** The *orientation* of a line segment e is the angle the line through e makes with the x-axis.

C.2 Preprocessing. By Theorem B.1, we can obtain a c-cluster tree T_c for P in linear time, given $\mathrm{DT}(P)$. Thus, we only need to construct the regular quadtrees T_u^Q for each node u in T_c. This is done by processing each node of T_c individually, but first we need to perform some preprocessing in order to assign the edges from $\mathrm{DT}(P)$ to the nodes of T_c. For every node $u \in T_c$, we define $\texttt{out}(u)$ as the set of edges in $\mathrm{DT}(P)$ that have exactly one endpoint in P_u and both endpoints in $P_{\bar{u}}$. Clearly, every edge is contained in exactly two sets $\texttt{out}(u)$ and $\texttt{out}(v)$, where u and v are siblings in T_c. The following is a simple variant of a lemma from KL [40, Lemma 3].

Lemma C.1. (Krznaric-Levcopolous) *Let P be a planar n-point set. Given $DT(P)$ and a c-cluster tree T_c for P, the sets $\texttt{out}(u)$ for every node $u \in T_c$ can be found in overall $O(n)$ time and space on a pointer machine.*

Proof. Our proof is almost identical to the one in KL, with the only difference that we invoke the off-line least-common-ancestor (lca) algorithm by Buchsbaum *et al.* [11] instead of the online algorithm by Harel and Tarjan [36] to obtain a linear-time algorithm for the pointer machine. More precisely, KL describe how to transform T_c in linear time into two binary trees T_L and T_R with $O(n)$ nodes such that the following holds: given an edge pq of $\mathrm{DT}(P)$, we can find the two nodes $u, v \in T_c$ with $pq \in \texttt{out}(u), \texttt{out}(v)$ by performing one lca-query each in T_L and T_R. Thus, the sets $\texttt{out}(u)$ can be found by $O(n)$ off-line lca queries in T_L and T_R, and using the result by Buchsbaum *et al.* [11, Theorem 6.1], this takes $O(n)$ time and space on a pointer machine. □

C.3 Processing a Single Node of T_c. Now let u be a node in T_c, and let $v_1, v_2, \ldots, v_m$ be the children of u. For each child v_i, let $\delta_i := d(P_{v_i}, P_u \setminus P_{v_i})$.

Claim C.2. *For $i = 1, \ldots, m$, $\texttt{out}(v_i)$ contains an edge of length δ_i.*

Proof. Note that if $\mathrm{DT}(P)$ contains an edge e incident to P_{v_i} with length δ_i, then e must be in $\texttt{out}(v_i)$, by the definition of a c-cluster. Since $\mathrm{emst}(P)$ is a subgraph of $\mathrm{DT}(P)$, it thus suffices to show that $\mathrm{emst}(P)$ contains such an edge. Consider running Kruskal's MST algorithm on P. According to the definition of a c-cluster, by the time the algorithm considers the edge e that achieves δ_i, the partially constructed EMST contains exactly one connected component that has precisely the points in P_{v_i}. Therefore, $e \in \mathrm{emst}(P)$, and the claim follows. □

Initialization. By scanning the sets $\texttt{out}(v_i)$, determine a child v_j with minimum δ_j (by Claim C.2 a shortest edge in $\texttt{out}(v_i)$ has length δ_i). We may assume that $j = 1$. Let S_1 be a square that contains P_{v_1} and that has side-length $\delta_1/8$. Let α be the smallest integer such that four squares of size $2^{\alpha-1}\delta_1/8$ cover all of P_u. Lemma B.3 implies that $\alpha = O(m)$.

The goal is to compute T_u^Q, the balanced regular quadtree aligned at S_1 such that each P_{v_i} is contained in squares of size $\delta_i/8$. To begin, we use S_1 to initialize T_u^Q as the partial balanced quadtree T_u^Q shown in Figure 9. Every square S of T_u^Q stores the following fields:

- $\texttt{parent}$: a pointer to the parent square, $\texttt{nil}$ for the root;
- $\texttt{children}$: pointers for the four children of S, $\texttt{nil}$ for a leaf;
- $\texttt{neighbors}$: links to the four orthogonal neighbors of S in the quadtree T_u^Q with size $|S|$ (or size $2|S|$, if no smaller neighbor exists);
- $\texttt{characteristic}$: a list of the at most eight *characteristic* edges of S, that is, the i-th entry in the list $\texttt{characteristic}$ contains a shortest edge

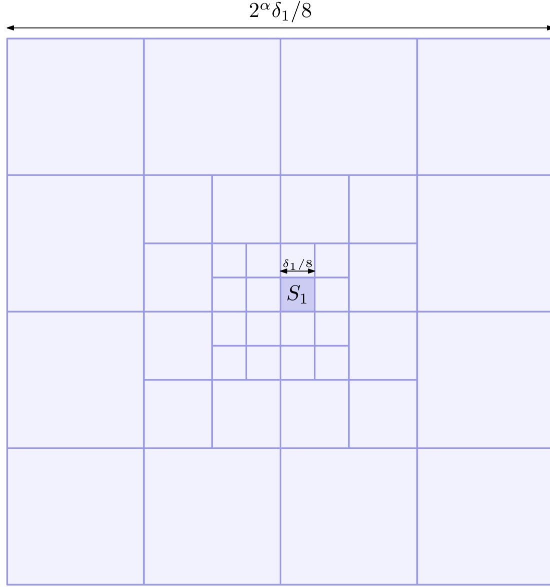

Figure 9: The initial quadtree.

e_i in $\mathrm{DT}(P)$ such that (i) e has exactly one endpoint inside S; (ii) e has one endpoint outside the neighborhood of S; and (iii) e has orientation in $[i\pi/4, (i+1)\pi/4)$;

- `shortcuts`: a list of *shortcuts*, one for each side of S.

The fields `parent`, `children`, and `neighbors` are initialized for all the nodes in T_Q; `characteristic` is initialized only for S_1, by scanning $\texttt{out}(v_1)$. `shortcuts` is initialized to be empty. The precise function of the lists `characteristic` and `shortcuts` is described in KL [40], and we will not discuss them much further here.

LEMMA C.3. *The total time for the initialization phase is* $O(m + \sum_{i=1}^m |\texttt{out}(v_i)|)$.

Proof. By Lemma B.3, the initial size of T_u^Q is $O(m)$. All other operations consist of scanning the `out`-lists or are linear in the size of T_u^Q. □

C.4 Building the Tree T_u^Q. Now we build the tree T_u^Q by a traversing $\mathrm{DT}(P)$ in a way reminiscent of Dijkstra's algorithm [23]. Refer to Algorithm 2. The algorithm is started by calling $\texttt{explore}(\{S_1\}, 2^{\alpha-1}\delta_1/8)$, and it proceeds according to the levels of T_u^Q. At each point, it maintains a set `active` of squares that contain a cluster that has already been processed, and it uses a function `findStar` to discover new clusters whose distance from the active clusters is comparable to the size of the squares in the current level. We will say

Algorithm 2 Computing a c-cluster quadtree for the children of a c-cluster.

$\texttt{explore}(\mathcal{S}, \texttt{maxsize})$

1. Set $\texttt{active} := \mathcal{S}$.
2. Set $\texttt{newActive} := \emptyset$.
3. Until the squares in `active` have size greater than `maxsize`:
   (a) For every square S in `active` call the function $\texttt{findStar}(S)$ to determine $\bigstar(S)$. Append $\bar{S}$ to `newActive`, if it is not present yet.
   (b) For every edge $e \in \bigcup_{S\in\texttt{active}} \bigstar(S)$, if e has an endpoint in an undiscovered cluster, call the function $\texttt{newCluster}(S, e)$, and append all the squares returned by this call to `newActive`.
   (c) Set $\texttt{active} := \texttt{newActive}$.

$\texttt{newCluster}(S, e)$

1. Walk along e through the current T_u^Q to find the square S' of T_u^Q that contains the other endpoint of e. This tracing is done by following the appropriate `neighbor` pointers from S.
2. Refine T_u^Q for the new cluster, and let $\mathcal{S}'$ be the set of leaf squares containing the newly discovered cluster.
3. Call $\texttt{explore}(\mathcal{S}', \text{size of squares in } \texttt{active})$. Afterwards, return the `active` squares from the recursive call.

more about `findStar` below. For each new cluster, we call `newCluster` which refines T_u^Q to accommodate the new cluster and recursively explores the short edges out of this new cluster. We give the details for the refinement in Step 2 of `newCluster`: Let v_j be the cluster that contains the other endpoint q of e (we can find v_j in constant time, since $e \in \texttt{out}(v_j)$, and since for each edge we store the two clusters whose `out`-lists contain it). Subdivide the current leaf square containing q (and possibly also its neighbors if they contain points from P_{v_j}) in quadtree-fashion until P_{v_j} is contained in squares of size $\delta_j/8$. Then balance the quadtree and update the `neighbor` pointers accordingly. Scan $\texttt{out}(v_j)$ in order to initialize `characteristic` for the leaf squares containing points in P_{v_j}.

The algorithm is recursive, and at each point there exists a sequence $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_z$ of instantiations of `explore`, where $\mathcal{E}_{i+1}$ was invoked by $\mathcal{E}_i$. Each $\mathcal{E}_i$ has a set $\texttt{active}_i$ of active squares, such that all squares

in each $\texttt{active}_i$ have the same size, and such that the squares in $\texttt{active}_{i+1}$ are not larger than the squares in $\texttt{active}_i$. We say that a square is active if it is contained in $\texttt{active}^T := \bigcup_i \texttt{active}_i$. The neighborhood of $\texttt{active}^T$ is the union of the neighborhoods of all boxes in $\texttt{active}$. We maintain the following invariant:

Invariant C.4. *At all times during the execution of* $\texttt{explore}$*, all undiscovered* c*-clusters lie outside the neighborhood of* $\texttt{active}^T$*.*

Claim C.5. *Invariant C.4 is maintained by* $\texttt{explore}$*.*

Proof. The set $\texttt{active}^T$ only changes in Steps 1 and 3c. The invariant is maintained in Step 1, since the size of the squares in $\mathcal{S}$ is chosen such that their neighborhoods contain no points from any other cluster.

Let us now consider Step 3c. The set $\texttt{newActive}$ contains two kinds of squares: (i) the parents of squares processed in the current iteration of the main loop; and (ii) squares that were added to $\texttt{newActive}$ after a recursive call. We only need to focus on squares of type (i), since squares of type (ii) are already added to $\texttt{active}^T$ during the recursive call. Suppose that $\texttt{active}^T$ contains a square S whose neighborhood has a point $p \in P$ in an undiscovered cluster. Since $S \in \texttt{active}^T$, there is a point $q \in P \cap \mathcal{S}$, and by the definition of neighborhood, we have $d(p, q) \leq 3|S|$. However, since the dilation of $\mathrm{DT}(P)$ is at most 2.5 [37], $\mathrm{DT}(P)$ contains a path π of length at most $8|S|$ from p to q. Let p' be the last discovered point along π. The point p' lies in an active square S' with $|S'| \geq |S|$, and the edge e leaving p' on π has length at most $8|S'|$. Therefore, $e \in \bigstar(S'')$ for a descendant S'' of S', which contradicts the fact that p' is the last discovered point along π. □

Lemma C.6. *The total running time of* $\texttt{explore}$*, excluding the calls to* $\texttt{findStar}$*, is* $O(m+\sum_{i=1}^{m} |\texttt{out}(v_i)|)$*.*

Proof. All squares appearing in $\texttt{active}^T$ are ancestors of non-empty leaf squares in the final tree T_u^Q. Therefore, by Lemma B.3, the total number of iterations for the loop in Step 3a is $O(m)$. Furthermore, $\bigstar(S)$ contains only edges of length $\Theta(|S|)$, so every edge appears in only constantly many stars. It follows that the total size of the $\bigstar$-lists, and hence the total number of iterations of the loop in Step 3b is $O(\sum_{i=1}^{m} |\texttt{out}(v_i)|)$.

It remains to bound the time for tracing the edges and balancing the tree. Since T_u^Q is balanced and since $\bigstar(S)$ contains only edges of length $\Theta(|S|)$, the tracing along the $\texttt{neighbor}$ pointers of an edge takes constant time (since we traverse constantly many boxes of size $\Theta(|S|)$). By Invariant C.4, the other endpoint of the edge is contained in a leaf square of the current T_u^Q of size $\Theta(|S|)$ (this is because the quadtree is balanced and because the other endpoint of the edge lies outside the neighborhood of the active squares). Therefore, the time to build the balanced quadtree for the new leaf squares containing the newly discovered cluster can be charged to the corresponding nodes in the final T_u^Q, of which there are $O(m)$ many. Furthermore, note that by Invariant C.4, balancing the quadtree for the newly discovered leaf squares does not affect any descendants of the active squares. □

C.5 Implementing $\texttt{findStar}$. KL show how to exploit the geometric properties of the Delaunay triangulation in order to implement the function $\texttt{findStar}$, quickly, by appropriately maintaining the $\texttt{characteristic}$ and $\texttt{shortcuts}$ lists of the active squares [40, Section 6]. This part of the algorithm works on a real RAM/pointer machine without any further modifications, so we just state their result.

Lemma C.7. *The total time for all calls to* $\texttt{findStar}$ *is* $O(m + \sum_{i=1}^{m} |\texttt{out}(v_i)|)$*.* □

C.6 Putting Everything Together. We can now finally prove Theorem 3.1.

Proof. (of Theorem 3.1) First, we use Theorem B.1 to find a c-cluster tree T_c for P in $O(n)$ time. Next, we use the algorithm from Section C.2 to preprocess the tree. By Lemma C.1, this also takes $O(n)$ time. Finally, we process each node of T_c using the algorithm from Section C.3. By Lemmas C.3, C.6, and C.7, this takes total time $\sum_j 1 + |\texttt{out}(v_j)|$, where the sum ranges over all the nodes of T_c. This sum is $O(n)$ because there are $O(n)$ nodes in T_c, and because every edge of $\mathrm{DT}(P)$ appears in exactly two $\texttt{out}$-lists. Hence, the total running time is linear, as claimed. □