A Replacement for Voronoi Diagrams of Near Linear Size*

Sariel Har-Peled†

June 11, 2001

Abstract

For a set P of n points in $\mathbb{R}^d$, we define a new type of space decomposition. The new diagram provides an ε-approximation to the distance function associated with the Voronoi diagram of P, while being of near linear size, for $d \geq 2$. This contrasts with the standard Voronoi diagram that has $\Omega\left(n^{\lceil d/2 \rceil}\right)$ complexity in the worst case.

1 Introduction

Voronoi diagrams are a fundamental structure in geometric computing. They are being widely used in clustering, learning, mesh generation, graphics, curve and surface reconstruction, and other applications. While Voronoi diagrams (and their dual structure Delaunay triangulations) are simple, elegant and can be constructed by (relatively) simple algorithms, in the worst case their complexity is $\Theta(n^{\lceil d/2 \rceil})$. Even in three dimensions their complexity can be quadratic and although such behavior is rarely seen in practice (for $d = 3$) this is not theoretically satisfying. Despite the vast literature on Voronoi diagrams [Aur91], they are still not well understood, and several major questions remain open (i.e., the complexity of the Voronoi diagram of lines in 3D [AS99]). Recently, there was effort to quantifying situations when the complexity of the Voronoi diagram in 3D has low complexity [AB01], and when it has high complexity [Eri01].

A Voronoi diagram is induced by the distance function of a point-set $P \subseteq \mathbb{R}^d$. The NN (nearest neighbor) distance function $V_P(q)$ returns the distance between q and its nearest point in P. The Voronoi diagram is the decomposition of $\mathbb{R}^d$ into cells, so that inside each cell c there is a single point $p_c \in P$ such that for any point $q \in c$ the nearest-neighbor of q in P is p_c.

In this paper, we consider the question of whether one can find an approximate Voronoi diagram of points in $\mathbb{R}^d$ of near linear size. In particular, we are interested in computing a decomposition of $\mathbb{R}^d$ into cells, so that each cell c has an associated point $p_c \in P$, such

*The updated version of this paper is available from the author web-page at [Har01]

†Department of Computer Science, DCL 2111; University of Illinois; 1304 West Springfield Ave.; Urbana, IL 61801; USA; http://www.uiuc.edu/~sariel/; sariel@uiuc.edu

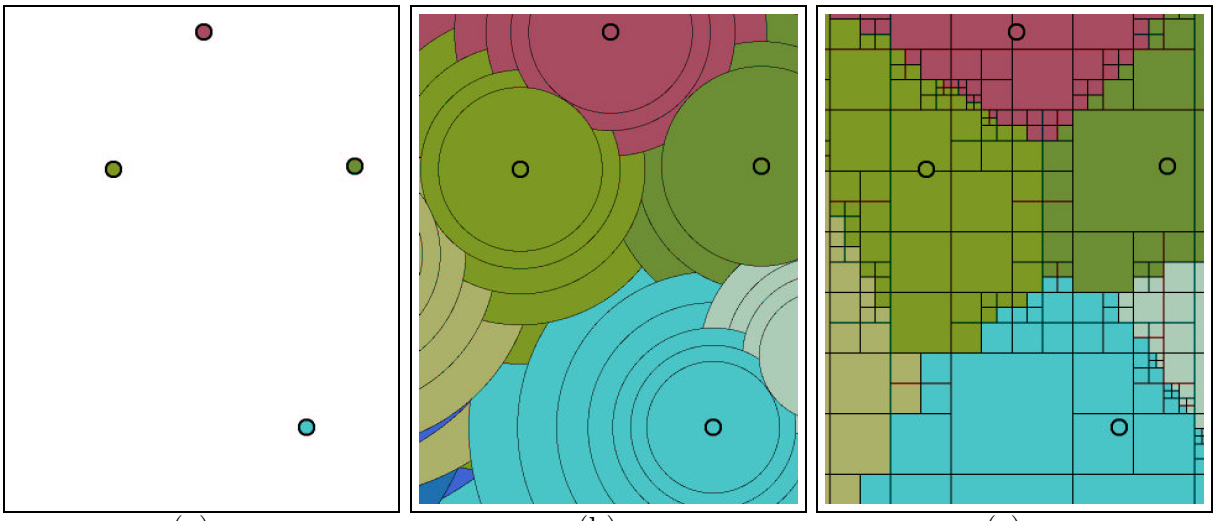

Figure 1: (a) The point-set. (b) The generated set of balls, such that answering approximate NN query is equivalent to performing a point-location query among them. (c) Streaming the balls to a compressed quadtree results in the required approximate Voronoi diagram.

that for any point in c the point p_c is an approximate NN in P. Overall, one would like to minimize the number of cells and the complexity of each cell in such a decomposition.

If one is interested only in ε-approximating the distance function $V_P(\cdot)$ on $\mathbb{R}^d$, then this can be done in near linear preprocessing time (and space) [Cla94, Cha98, AMN+98], and the query time is polynomial in $\log n$ and $1/\varepsilon$. However, those data-structures suffer from exponential dependency on the dimension. Recently Indyk and Motwani [IM98] and Kushilevitz *et al.* [KOR98] have presented data-structures of polynomial size and query time with polynomial dependency on the dimension. However, all those data-structures do not have a space decomposition associated with them. Clarkson construction [Cla94] can be interpreted as inducing a covering of space by approximate cells (of the Voronoi cells), and it answers NN queries by performing a sequence of point-location queries in those cells. However, those cells are not disjoint in their interior. In fact, if one is willing to compromise on a space covering instead of space decomposition the problem becomes considerably easier.

In this paper, we present the following results:

- A space decomposition that ε-approximate the Voronoi diagram. It is made out of $O\left(\frac{n}{\varepsilon^d}(\log n)\log\frac{n}{\varepsilon}\right)$ cells. Each cell is either a cube or the difference between two cubes. The cells are the regions arising from an appropriate compressed quadtree.

- Using this compressed quadtree one can answer NN-queries in $O(\log(n/\varepsilon))$ time. This is considerably faster than any previous data-structures, as it is the first one that have logarithmic dependency in the query time on both n and ε. The previously fastest query time is achieved by Indyk and Motwani [IM98]. Their construction is inferior to ours in all relevant parameters (preprocessing time, space, query time) by a factor polynomial in $1/\varepsilon, \log n$.[1]

[1]Unfortunately, Indyk and Motwani do not explicitly state their bounds, and we are unable to provide

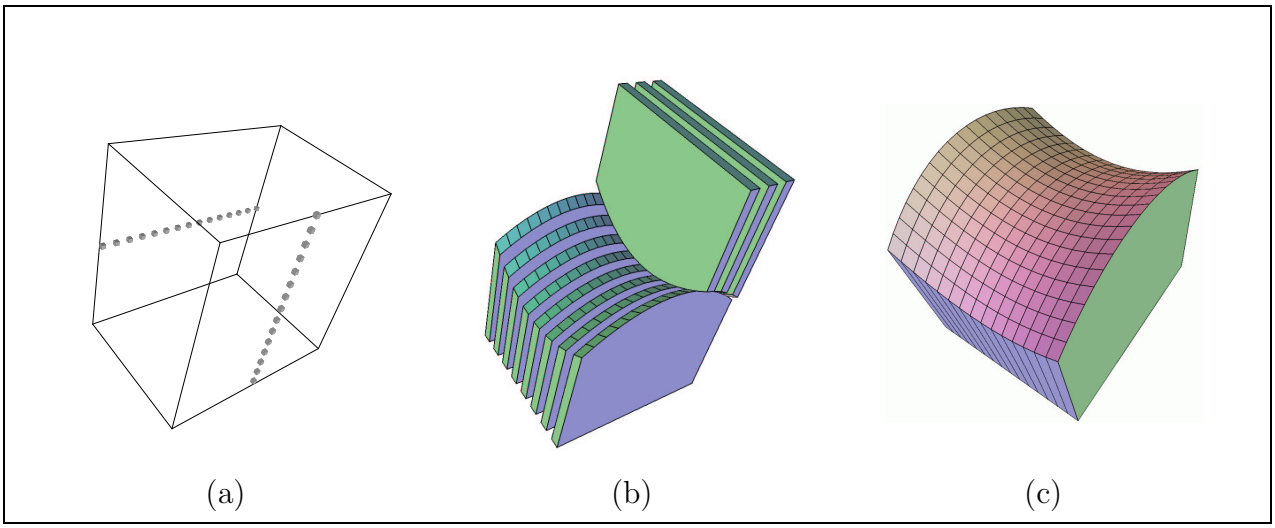

Figure 2: (a) The point-set in 3D inducting a Voronoi diagram of quadratic complexity. (b) Some cells in this Voronoi diagram. Note that the cells are thin and flat, and every cell from the lower part touches the cells on the upper part. (c) The contact surface between the two parts of the Voronoi diagram has quadratic complexity, and thus the Voronoi diagram itself has quadratic complexity.

- In the planar case, one can extract a fat triangulation from the quadtree that ε-approximates the Voronoi diagram. It is made out of $O\left(\frac{n}{\varepsilon^2}\log\frac{\rho(P)}{\varepsilon}\right)$ triangles, where $\rho(P)$, the spread of P, is the ratio between the diameter and the distance between the closest pair. Since the overlay of two fat triangulations has linear complexity, this is useful for applications requiring the overlay of the Voronoi diagram with other structures.

  Similar bounds hold for decomposition of space into fat simplices.

- To achieve the above results, we describe a relatively simple reduction from NN queries to point-location in equal balls (PLEB) queries (see details below). We outline the reduction in Section 2.1. Although such a reduction was previously shown by Indyk and Motwani [IM98], our reduction is considerably simpler, more intuitive, more efficient in the number of PLEB queries required to answer NN queries, the overall space used, and the construction time.

General Idea The idea underlying the construction of the approximate Voronoi diagram arises from inspecting the standard example of a quadratic complexity Voronoi diagram in 3D depicted in Figure 2 (a). This point-set consists of two collections of points placed along two segments in 3D. In this construction, there are $\Omega(n^2)$ pairs of cells that have a common boundary. However, those cells are very thin slices, and it clear that far enough from their respective sites, it is possible to shrink portions of one cell, while increasing the size of an another adjacent cell in order to minimize interaction between cells, while preserving the

direct comparison with their data-structure

approximate NN property.

To this end, we consider the construction of the Voronoi diagram to be a tournament between the sites for regions of influence. Namely, inflate a ball around each point. Those balls inflate with the same speed around each point, and a point x belongs to site p, if the ball of p is the first ball to contain x.

Now, consider two sites x and y. Clearly, if the radi of the balls around x and y are large enough, then we need to grow only one of those balls, as from this distance onward x and y are almost equivalent. See Figure 3. Assume, that x lost and from this point on the ball of x does not grow anymore while the ball around y continues to grow. Thus, the cell of x is now fixed, as x dropped out of the tournament. Namely, the cell of x is now smaller than its (exact) Voronoi cell, and intuitively, it would not interact with other cells far from it.

We implement this tournament mechanism indirectly, by first constructing a hierarchical clustering of the sites, and then using this clustering to build a data-structure for answering NN queries. Finally, we extract from this NN data-structure the relevant set of "critical" balls needed to perform this tournament. Having this set of (prioritized) balls, it is now possible to generate the approximate Voronoi diagram.

In Section 2.2 we describe how to hierarchically cluster the input points into a tree (this is a variant of nearest-neighbor clustering [DHS01]), and give a fast algorithm for computing this clustering. Next, in Section 2, we use this clustering to reduce the NN search problem to a sequence of point-location queries among equal balls. In Section 3, we flatten this construction showing how to answer NN queries by doing point-location among balls. We present the resulting algorithm and applications in Section 4. Concluding remarks are given in Section 5.

2 NN Search via PLEB

2.1 Outline

In this section, we outline the reduction from NN search to a sequence of point-location queries among equal balls (i.e., PLEB queries). The detailed (and somewhat messy) construction is described in Section 2.3.

For a point-set P with n points and a parameter r, let $\mathbb{UB}_P(r) = \cup_{p \in P} B(p, r)$ denote the union of equal balls of radius r centered at the points of P.

Definition 2.1 For a point-set P in $\mathbb{R}^d$, and $q \in \mathbb{R}^d$, let $\mathcal{NN}_P(q)$ denote the point of P which is closest to q, let $d_P(q) = \|q\mathcal{NN}_P(q)\|$. For a parameter $\gamma > 0$, $y \in P$ is an *γ-approximate nearest neighbor* of q (i.e., γ-NN) if $d_P(q) \leq \|qy\| \leq (1+\gamma)d_P(q)$.

Definition 2.2 Given a set of points P in $\mathbb{R}^d$, and a parameter r, a PLEB(P, r) (*point-location in equal balls*) is a data-structure so that given a query point $q \in \mathbb{R}^d$, it decides whether $q \in \mathbb{UB}_P(r)$. Furthermore, if $q \in \mathbb{UB}_P(r)$, then it returns a point u of P such that $q \in B(u, r)$.

For a prespecified radius r^*, and a query point q, one can decide whether $d_P(q) \leq r^*$ by checking if $q \in \mathbb{UB}_P(r^*)$, where $d_P(q)$ is the distance of q to its nearest neighbor in P. This can be carried out by constructing $\mathcal{P} = \text{PLEB}(P, r^*)$, and performing a PLEB query in $\mathcal{P}$.

Let r_{median} be the radius for which $\mathbb{UB}_{med} = \mathbb{UB}_P(r_{median})$ has exactly $n/2$ connected components. By the above discussion, we can construct a PLEB $\mathcal{P}_{med} = \text{PLEB}(P, r_{median})$. Given a query point q, if $q \in \mathbb{UB}_{med}$ (this can be decided by one point-location query in $\mathcal{P}_{med}$), then we continue the search for the NN recursively in the connected component of $\mathbb{UB}_{med}$ that contains q. Since $\mathbb{UB}_{med}$ have $n/2$ connected components, it follows, that the search continues recursively into a subset of P of cardinality at most $1 + n/2$.

Alternatively, if $d_P(q) \geq r_{top} = (4nr_{median} \log n)/\varepsilon$ (this can be decided by a single PLEB query in $\text{PLEB}(P, r_{top})$), then q is "faraway" from the points of P, and we can continue the search on a decimated subset of P. Namely, from each connected of $\mathbb{UB}_{med}$, we extract one point of P that lies inside it. This results in a set $P^+ \subset P$ that contains $n/2$ points. We continue the recursive search into P^+. Although continuing the search int P^+ introduces accumulative error into the search results, one can argue that overall the error introduces is smaller than $1 + \varepsilon/3$.

The only case that remains unresolved, is when $r_{median} \leq d_P(q) \leq r_{top}$. Observe, that $r_{top}/r_{median} = O((n \log n)/\varepsilon)$. Namely, we can cover the interval $[r_{median}, r_{top}]$ by $M = \log_{1+\varepsilon/3} r_{top}/r_{median} = O((1/\varepsilon) \log (n/\varepsilon))$ PLEBs: $\mathcal{P}_1, \ldots, \mathcal{P}_M$, where

$$\mathcal{P}_i = \text{PLEB}(P, r_{median}(1 + \varepsilon/3)^i),$$

for $i = 1, \ldots, M$.

By performing a binary search on $\mathcal{P}_1, \ldots, \mathcal{P}_M$ one can find, using $O(\log M)$ PLEB queries, the index i, such that $q \notin \mathbb{UB}(P, r_{median}(1 + \varepsilon/3)^i)$ and $q \in \mathbb{UB}(P, r_{median}(1 + \varepsilon/3)^{i+1})$. Namely, we had found an ε-NN to q in P.

Overall, given a query point, either we found its ε-NN using $O(\log M) = O(\log (n/\varepsilon))$ PLEB queries, or alternatively, we performed two PLEB queries and continued the search recursively into a set having at most $n/2 + 1$ points of P. Thus, one can find an ε-NN to a point by performing $O(\log (n/\varepsilon))$ PLEB queries.

2.2 Hierarchical Clustering

In the following, let P be a set of n points in $\mathbb{R}^d$.

Lemma 2.3 *Let $x, y, q \in \mathbb{R}^d$, and $\gamma \geq 0$ a parameter such that $\|xy\| \leq \gamma \|qx\|$ and $\|xy\| \leq \gamma \|qy\|$. Then $\|qx\| \leq (1 + \gamma) \|qy\|$, and $\|qy\| \leq (1 + \gamma) \|qx\|$.*

Proof: $\|qx\| \leq \|qy\| + \|yx\| \leq \|qy\| + \gamma \|qy\| = (1 + \gamma) \|qy\|$.

The other claim follows by symmetry. ∎

Let q be a query point, and we would like to find a γ-NN to q in P. Lemma 2.3 tell us that if we have two points that are both "far enough" from q and relatively close together, than we can consider only one of them. Namely, if one has a lower bound on $d_P(x)$, then one can throw away points from P which are close together (i.e., preserving only one representative out of such a cluster of points) and consider only the remaining points in the NN query process.

2.2.1 Constructing approximate minimum spanning tree

In the following, we need the MST (minimum spanning tree) to cluster the points hierarchically. Unfortunately, for $d > 2$, no near-linear time algorithm is known for this problem. Instead, we satisfy ourselves with an approximation to the MST.

Definition 2.4 A tree T having the points of P in its nodes, is an *λ-MST* of P if it is a spanning tree of P, having a value $\mathrm{len}(\cdot)$ associated with each edge of T, such that for any edge of $e = uv \in T$, we have that $\mathrm{len}(e) \leq d(P_1, P_2) \leq \lambda\,\mathrm{len}(e)$, where P_1, P_2 is the partition of P into two sets as induced by the two connected components of $T \setminus \{e\}$, and $d(P_1, P_2) = \min_{x \in P_1, y \in P_2} \|xy\|$ is the *distance* between P_1 and P_2.

Remark 2.5 Note, that the above definition of approximate MST diverges from the regular definition: We do not care about the overall weight, but rather, we insist that the weight of every edge would be not to far from the true weight in the MST of the point-set.

Lemma 2.6 *One can compute a 2-MST Λ of P in $O_d(n \log n)$ time, where $O_d(\cdot)$ hides a constant that depends exponentially on d.*

Proof: We use spanners, as generated by the WSPD (well separated pairs decomposition) construction of Callahan and Kosaraju [CK95]. Namely, in time $O_d(n \log n + n/\varepsilon^d)$ we compute a spanner (i.e., sparse graph) G over the points of P with $O_d(n/\varepsilon^d)$ edges, so that the distance between any two points of P in G is a $(1+\varepsilon)$ approximation to their euclidean distance. Setting $\varepsilon = 1$, results in a running time $O_d(n \log n)$. The MST Λ of G is the required spanning tree.

If an edge $e \in E(G)$ is assigned weight w, then we set $\mathrm{len}(e) = w/2$. It is easy to verify that this results in a 2-MST. ∎

If the dimension is not a small constant, then the algorithm Lemma 2.6 is too slow to be used, as it has an exponential dependency on the dimension.

Lemma 2.7 *One can compute a nd-MST of P in $O(nd \log^2 n)$ time.*

Proof: The construction is similar to the construction of the fair-split tree of Callahan and Kosaraju [CK95] and we only sketch the algorithm. Our construction is based on a recursive decomposition of the point-set. In each stage, we split the point-set into two subsets. We recursively compute a nd-MST for each point-set, and we merge the two trees into a single tree, by adding an edge, and assigning it appropriate weight. To carry this out, we try to separate the set into two subsets that are furthest away from each other.

Let $R = R(P)$ be the minimum axis parallel box containing P, and let $l = l(P) = \sum_{i=1}^{d} \|I_i(R)\|$, where $I_i(R)$ is the projection of R to the i-th dimension.

Clearly, one can find an axis parallel strip H of width $\geq l/((n-1)d)$, such that there is at least one point of P on each of its sides, and there is no points of P inside H. Indeed, to find this strip, project the point-set into the i-th dimension, and find the longest interval between two consecutive points. Repeat this process for $i = 1, \ldots, d$, and use the longest interval encountered. Clearly, the strip H corresponding to this interval is of width $\geq l/((n-1)d)$.

Now recursively continue the construction of two trees T^+, T^-, for P^+, P^-, respectively, where P^+, P^- is the splitting of P into two sets by H. Pick any two points $p \in P^+, q \in P^-$,

create the tree $T = T^+ \cup T^- \cup \{e\}$, where $e = pq$. Finally, set $\mathrm{len}(e) = l/((n-1)d)$. Clearly, $\mathrm{len}(e) \leq d(P^+, P^-) \leq l(P) = nd \cdot \mathrm{len}(e)$.

The construction can be performed in $O(nd \log^2 n)$ time using the techniques of Callahan and Kosaraju [CK95] and the straightforward details are omitted.

We claim that the resulting tree is a nd-MST. Indeed, if the current set being handled is Q, then $l(Q^+), l(Q^-) \leq l(Q)$, where Q^+, Q^- are the splitting computed of Q into two subsets. Let e_Q be the edge added to the resulting tree Λ, connecting an arbitrary point of Q^+ to an arbitrary point of Q^-.

In this stage, we found a partition of Q into Q^+, Q^- of distance $\geq l(Q)/(nd)$ from each other. Furthermore, we know by induction that $d(Q, P \setminus Q) \leq l(\mathrm{parent}(P))/nd \geq l(Q)/nd$. Thus,

$$\begin{aligned} l(Q) \geq d(Q^+, P \setminus Q^+) &= \min(d(Q^+, Q^-), d(Q^+, P \setminus Q)) \\ &\geq \min\left(\frac{l(Q)}{nd}, d(Q, P \setminus Q)\right) \geq \min\left(\frac{l(Q)}{nd}, \frac{l(\mathrm{parent}(Q))}{nd}\right) \\ &\geq \frac{l(Q)}{nd} = \mathrm{len}(e_Q), \end{aligned}$$

as $l(\mathrm{parent}(Q)) \geq l(Q)$. A similar argument holds for Q^-. Namely, the edge e_Q in the spanning tree connecting the sets Q^+ and Q^- is indeed nd-approximation to the distance of Q^+, Q^- from the rest of the point-set. ∎

2.2.2 Computing λ-stretch hierarchical clustering

Assume that we are given a λ-MST T, we now use it to cluster the points. For a parameter r, let $\mathbb{UB}_P(r) = \cup_{p \in P} B(p, r)$, where $B(p, r)$ is the ball of radius r centered at p. $\mathbb{UB}_P(r)$ is a monotone growing set such that $\mathbb{UB}_P(+\infty) = \mathbb{R}^d$. In particular, the distance between x and the closest point of P, is the value of r for which $\mathbb{UB}_P(r)$ has x on its boundary. Namely, we sweep space by the balls centered at the points of P.

Let $\mathcal{U}_P(r)$ denote the set of connected components of $\mathbb{UB}_P(r)$, and let $\mathfrak{CC}_P(r) = \left\{ P \cap X \;\middle|\; X \in \mathcal{U}_P(r) \right\}$ be the partition of P into the connected components of $\mathbb{UB}_P(r)$. We would like to trace the history of $\mathfrak{CC}_P(r)$ as r increases from $r = 0$ to $r = +\infty$. We define a tree as follows: In the beginning, it is a forest (corresponding to $\mathfrak{CC}_P(0)$) where each point of P is a singleton. Every time $\mathfrak{CC}_P(r)$ changes, we have to merge two trees (i.e., the boundary of two different connected components of $\mathcal{U}_P(P)$ collided). We hang the smaller tree on the root of the larger tree (if the two trees are of equal size we resolve this in an arbitrary fashion). Let $G(r)$ denote this forest at time r. Note, that a tree of $G(r)$ corresponds to a connected component of $\mathbb{UB}_P(r)$. Let $\mathcal{G} = G(\infty)$ denote the resulting tree.

Computing $\mathcal{G}$ seems to be expensive, as it is equivalent to computing the minimum spanning tree of P. We can view $\mathcal{G}$ as a hierarchical clustering of P. Instead, we construct an approximation to the hierarchical clustering of $\mathcal{G}$.

Definition 2.8 Let $\mathcal{F}$ be a directed tree storing the points of P in its nodes, so that for any point $p \in P$ there in as an associated value $r_{loss}(p)$. This value is also associated with the out-edge emanating from p in $\mathcal{F}$.

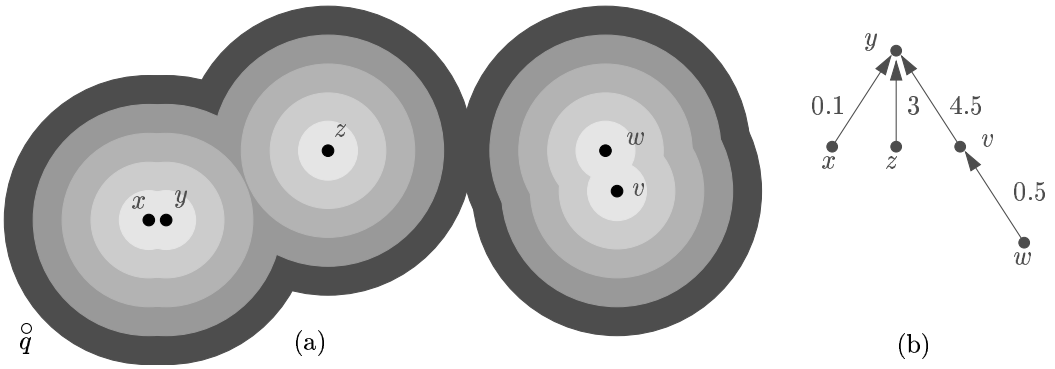

Figure 3: (a) A point-set and the union of balls as it evolves over time. Note that if a query point q is far enough from x and y it follows that its distance from x and y is almost identical. (b) The hierarchical clustering of the points.

Furthermore, let $\mathcal{F}(r)$ denote the forest generated from $\mathcal{F}$ by removing from it all the edges $e \in \mathcal{F}$ such that $r_{loss}(e) > r$. Let $\mathcal{CC}_P(r) = \mathcal{CC}_P^{\mathcal{F}}(r)$ be the partition of P into subsets as induced by the connected components of $\mathcal{F}(r)$.

$\mathcal{F}$ is a *λ-stretch hierarchical clustering of P*, if for any r, we have $\mathfrak{CC}_P(r) \sqsubseteq \mathcal{CC}_P(r) \sqsubseteq \mathfrak{CC}_P(\lambda r)$, where $\mathcal{A} \sqsubseteq \mathcal{B}$ if for any $X \in \mathcal{A}$, we have $Y \in \mathcal{B}$ such that $X \subseteq Y$ (i.e., one can compute $\mathcal{B}$ from $\mathcal{A}$ by a sequence of merges of sets of $\mathcal{A}$). Intuitively, this means that $\mathcal{F}$ is a λ-approximation to $\mathcal{G}$.

See Figure 3 (b) for an example of such a clustering.

Lemma 2.9 *One can compute a nd-stretch hierarchical clustering of P in $O(nd \log^2 n)$ time.*

Proof: We compute a nd-MST Λ of P in $O(nd \log^2 n)$ time, using the algorithm of Lemma 2.7. We now sort the edges of T by their len value. Next, we compute $\mathcal{F}$ by starting from a forest that corresponds to all the (singleton) points of P, and performing a sequence of merges.

In the i-th iteration, we take the i-th edge $x_i y_i$ and merge the two trees T_i^x, T_i^y that contains x_i and y_i, respectively. To do so, we create a new edge e_i connecting the two roots of T_i^x and T_i^y, and orient it an arbitrary fashion (i.e., we "hang" (say) T_i^x on T_i^y). Finally, we set $r_{loss}(e_i) = \text{len}(x_i y_i)/2$, where $\text{len}(x_i y_i)$ is the weight associated with the edge $x_i y_i$ in Λ.

It is easy to verify that the resulting tree $\mathcal{F}$ is a nd-stretch hierarchical clustering of P. ■

The tree $\mathcal{F}$ encode information about the shape and the NN metric induced by the point-set P. In the following, we further investigate this connection and show how to answer ε-NN queries using a structure computed using $\mathcal{F}$.

At any point in time, a point $p \in P$ is represented by $\text{rep}(p, r)$, which is the root of the tree of $\mathcal{F}(r)$ that contains p.

Lemma 2.10 *Let $p \in P$, and $q = \text{rep}(p, r)$, then $\|pq\| \leq 2nr\lambda$, where $\text{rep}(\cdot, \cdot)$ is defined by a given λ-stretch hierarchical clustering $\mathcal{F}$ of P.*

Proof: The points p and q belong to the same connected component $\mathcal{C}$ of $\mathbb{UB}_P(r\lambda)$. The radius of balls forming $\mathbb{UB}_P(r)$ is $r\lambda$, and the number of balls in $\mathcal{C}$ is bounded by n. ■ For a point $p \in P$, let $r_{loss}(p)$ be the minimum value of r for which $p \neq \mathrm{rep}(p, r)$; that is, the time where the cluster that p represents is being merged ("hanged") into a large cluster. This is the r_{loss} value associated with the directed edge emanating from p in $\mathcal{F}$.

The edges of $\mathcal{F}$ are directed. For a point $p \in P$, the edge emanating from it upward is generated at time $r_{loss}(p)$ (here, we view r as the time). This is the moment in time when p becomes dominated by $\mathrm{parent}(p)$.

2.3 Detailed construction

In the following, we assume that we are given $\mathcal{F}$, a λ-stretch clustering of P. This defines for each point $p \in P$ a value $r_{loss}(p)$.

Definition 2.11 For an approximation parameter γ, let $r_{death}(p) = (6\lambda r_{loss}(p) n \log n)/\gamma$.[2] For $p \in P$, we define the *transition time* of p to be the interval $I(p) = [r_{loss}(p), r_{death}(p)]$. Let x be a query point. If $d_P(x) \geq L$, then one can trim P as follows: We remove from P all the points that have $r_{death}(p) \leq L$, let $P(L)$ denote the resulting point-set.

Remark 2.12 The definition of r_{loss} and r_{death}, provides us now with the evolving set $P(r)$. Intuitively, what we got is a multi-resolution representation of the point-set P. This provides us with a natural way of decimating P when processing a point-location query q, if we ensure that $d_P(q)$ is large enough.

The data-structure we describe for answering approximate NN queries is performing a binary search for the value of $d_P(q)$, while using the accumulated information to decimate the point-set we search over and restricting the algorithm to the appropriate level of detail of P provided by $P(\cdot)$.

Lemma 2.13 *For a query point $q \in \mathbb{R}^d$ and a parameter L, let Q be a subset of P, such that Q induces a connected subtree of $\mathcal{F}$, $Q(L)$ is not empty, and $L \leq d_P(q) \leq d_Q(q) \leq (1+\alpha) d_P(q)$. Then $d_P(q) \leq d_{Q(L)}(q) \leq \left(1 + \frac{\gamma}{3 \log n}\right)(1+\alpha) d_P(q)$; namely, by restricting the NN search from Q to $Q(L)$, the quality of the approximation to the NN has deteriorated by at most a factor of $(1 + \gamma/(3 \log n))$.*

Proof: Let z be the NN of q in Q. If $z \in Q(L)$ we are done. Otherwise, let u be lowest ancestor of z in $\mathcal{F}$ which is in $Q(L)$. Let π be the path in $\mathcal{F}$ from z to u, and let w be the child of u in π. Namely, $\pi = z \to \cdots \to w \to u$.

Clearly, $r_{death}(w) \leq L$. Arguing as in the proof of Lemma 2.10, we have

$$\begin{aligned} \|uz\| &\leq 2n\lambda r_{loss}(w) \leq \frac{\gamma}{3\log n} r_{death}(w) \leq \frac{\gamma}{3\log n} L \leq \frac{\gamma}{3\log n} d_P(q) \\ &\leq \frac{\gamma}{3\log n} \min\left(\|uq\|, \|zq\|\right). \end{aligned}$$

[2] We use the convention that $\log n = \log_2 n$.

By Lemma 2.3, we have

$$\begin{aligned} d_{Q(L)}(q) &\leq \|uq\| \leq \left(1+\frac{\gamma}{3\log n}\right)\|qz\| = \left(1+\frac{\gamma}{3\log n}\right) d_Q(q) \\ &\leq \left(1+\frac{\gamma}{3\log n}\right)(1+\alpha) d_P(q). \end{aligned}$$

■ One can easily construct an IPLEB (interval PLEB), to answer PLEB queries over a range of distances.

Lemma 2.14 *For $1/2 > \gamma > 0$, given an interval $[a,b]$, and a point-set P, one can construct $O((\log b/a)/\gamma)$* PLEB*s, so that given a query-point q, one can decide if:*

(i) $d_P(q) \leq a$.

(ii) $d_P(q) \geq b$.

(iii) Find a point $y \in P$, so that $\|qy\| \leq (1+\gamma/3) d_P(q)$.

If (i) or (ii) happens only two PLEB *queries are being carried out, if (iii) happens then $O(\log((\log(b/a)/\gamma))$* PLEB *queries are being performed.*

Let $\mathcal{I}(P,a,b,\gamma)$ denote this interval PLEB.

Proof: Let $r_i = a(1+\gamma/3)^{i-1}$, for $i = 1,\ldots,M-1$, where $M = \left\lceil log_{(1+\gamma/3)}(b/a) \right\rceil = O\left(\frac{\log(b/a)}{\log(1+\gamma/3)}\right)$. Since $\gamma < 1/2$, by the Taylor expansion of $\ln(1+x)$, we have $M = O\left(\frac{\log(b/a)}{\gamma}\right)$. We set $r_M = b$. Let $\mathcal{P}_i = \text{PLEB}(P, r_i)$ for $i = 1,\ldots,M$.

Clearly, by one query to $\mathcal{P}_1$ we can decide if (i) happens (i.e., $q \notin \mathbb{UB}_P(r_1)$), and with one query to $\mathcal{P}_M$ we can decide if (ii) happens (i.e., $q \in \mathbb{UB}_P(r_m)$). Otherwise, (ii) must happen, and then $q \notin \mathbb{UB}_P(r_1)$ and $q \in \mathbb{UB}_P(r_M)$. Performing a binary search, on $\mathcal{P}_1,\ldots,P_M$, we can find the i such that $q \notin \mathbb{UB}_P(r_i)$ and $q \in \mathbb{UB}_P(r_{i+1})$.

Let y be the point of P that is returned by $\mathcal{P}_{i+1}$, because $q \in B(y, r_{i+1})$. Clearly, $r_i \leq d_P(q) \leq \|qy\| \leq r_{i+1} \leq (1+\gamma/3) r_i \leq (1+\gamma/3) d_P(q)$. Thus, y is an $\gamma/3$-NN of P, and we are done. ■

Definition 2.15 For $X \in \mathcal{CC}_P(r)$, let $\kappa(X)$ denote the point of X which is the root of the tree corresponding to X in $\mathcal{F}(r)$.

For a point-set $X = \{x_1,\ldots,x_m\} \subseteq P$, let $r_{median}(X)$ be the median value of $r_{loss}(x_1)$,… , $r_{loss}(x_m)$, and let $r_{top}(X) = ((36\lambda|X|\log n)/\gamma) r_{median}(X)$, where $n = |P|$.

Lemma 2.16 *For $X \in \mathcal{CC}_P(r_{median}(P))$ we have $P(r_{top}(P)) \cap X \subseteq \{\kappa(X)\}$.*

Proof: All the points of X, except maybe $\kappa(X)$, have $r_{loss} \leq r_{median}(P)$. In particular, for any $x \in X, x \neq \kappa(X)$, we have $r_{death}(x) \leq r_{top}(P)$, and thus they are not in $P(r_{top}(P))$. ■

The algorithm for constructing approximate NN search tree using PLEBs, `BuildNNTree`, is depicted in Figure 4. Although it stated as an algorithm on the point-set P, it is in fact a divide and conquer algorithm on the tree $\mathcal{F}$: In each stage, we break $\mathcal{F}$ into subtrees, and continue our search for the NN in the relevant subtree. See Figure 5.

```
FUNC BuildNNTree( F, M, γ, r^min, r^max )
   Input: F - nd-hierarchical clustering
                of M
          M ⊆ P - set of points
          γ - approx. parameter
          [r^min, r^max] - range of distances
              that should be considered
              for NN queries
   Output: Tree T_M for NN-search on M
begin
   Create a node v
   Set r_v^- ← max(r_median(M), r^min)
   Set r_v^+ ← min(r_top(M), r^max)
   Set P_v ← M
   if r_v^- ≥ r_v^+ then return nil.
   Compute I_v ← I(M, r_v^-, r_v^+, γ/3).
   M^+ ← M(r_v^+).
   T ← BuildNNTree( M^+, γ, r_v^+, r^max )
   Set T outer child of v: v_outer ← T.
   for X ∈ CC_M(r_v^-) do
      if |X| > 1 then
         v_X ← BuildNNTree( X, γ,
                   r^min, r_v^- )

   return v
end BuildNNTree
```

```
FUNC FindApproxNN( T, q )
   Input:  T - NN search tree
           q - query point
   Output: Approx. NN to q in P
begin
   v ← Root(T).

   Decide whether r_v^- ≤ d_P(q) ≤ r_v^+
         using Lemma 2.14 on I = I_v.
   if d_P(q) ≥ r_v^+ then
      return FindApproxNN( v_outer, q )

   Let u be the point returned by I
   if r_v^- ≤ d_P(q) ≤ r_v^+ then
      return u.

   (* d_P(q) ≤ r_v^-  *)
   Let X ∈ CC_P(r_v^-) s.t. u ∈ X
   if |X| = 1 then
      return u.

   return FindApproxNN( v_X, q)
end FindApproxNN
```

Figure 4: Construction of NN search tree. Note, that `BuildNNTree` uses the λ-stretch hierarchical clustering $\mathcal{F}$ of P in computing the r_{loss}, r_{death} values associated with each point of P. Those values are used in the computation of $r_{median}(M), r_{top}(M)$.

Lemma 2.17 *If* `BuildNNTree` *is called on a set $X \subseteq P$, then the induced subgraph of $\mathcal{F}$ on X is connected.*

Proof: Note, that the value of $r_{death}(\cdot)$ is monotone increasing along a path from any node of $\mathcal{F}$ to the root of $\mathcal{F}$. It follows, that the result always hold for $P(r_v^+)$.

As for the other recursive calls. Note that all the other recursive calls are on sets of $\mathcal{CC}_P(r_v^-)$, which are by definition connected subtrees of $\mathcal{F}$. ■

Lemma 2.18 *The depth of the tree constructed by* `BuildNNTree` *is at most $\lceil \log n \rceil + 1$, where $n = |P|$.*

Proof: Note, that all points of $p \in P$, such that $r_{loss}(p) \leq r_{median}(P)$, have $r_{death}(p) \leq r_{top}(P)$. Thus, at least $n/2$ of the points of P are not present in $P(r_{top}(P))$. Namely, $|P(r_{top}(P))| \leq |P|/2$.

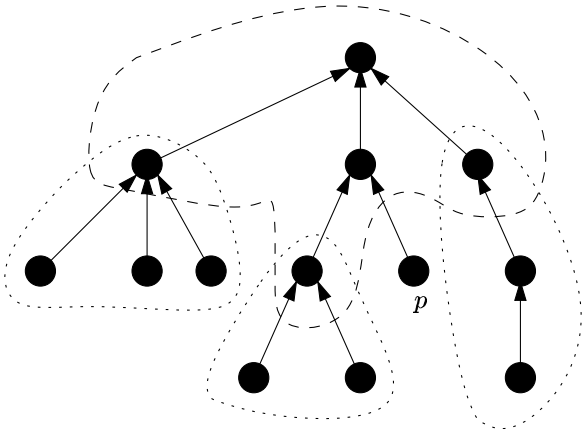

Figure 5: Given a point-set M, and its hierarchical clustering $\mathcal{F}$, the algorithm `BuildNNTree` constructs the search structure $\mathcal{T}_M$ by breaking $\mathcal{F}$ into several subtrees. The different subsets of M being created are depicted. Note that a point of M might participate in at most two subtrees: the cluster that it represents, and in the top subtree corresponding to the outer child of v. Note, that p being a subtree of a single point, does not require a recursive call to construct its own search subtree.

Similarly, since there are $n/2$ points of P with $r_{loss} > r_{median}(P)$, it follows that $\mathcal{U}_P(r_{median}(P))$ have at least $n/2$ connected components. Namely, each $X \in \mathbb{UB}_P(r_{median}(P))$ have cardinality at most $\leq n/2$. ■

Definition 2.19 For a node $v \in \mathcal{T}_P$, let $h(v)$ denote the *height* of the subtree of $\mathcal{T}_P$ stored at v, and let $n_v = |P_v|$.

Lemma 2.20 *The overall number of points stored in each level of* $\mathcal{T}_P =$ `BuildNNTree`(P, ε) *is* $3n$.

Proof: An edge $\overrightarrow{pq}$ of $\mathcal{F}$, has at most one node in each level of $\mathcal{T}_P$ that contains both p and q. Indeed, consider a node $v \in \mathcal{T}_P$ such that $p, q \in P_v$. If $r_{loss}(p) > r_v^-$, then p, q belong to two different sets in $\mathcal{CC}_{P_v}(r_v^-)$, and only $P_v(r_v^+)$ might contain both p and q. If $r_{loss}(p) < r_v^-$ then p, q belong to the same set in $\mathcal{CC}_{P_v}(r_v^-)$, but only q might appear in $P_v(r_v^+)$.

By Lemma 2.17, if a node v was created with a set $P(v)$, such that $|P_v| \geq 2$, than $P(v)$ form a connected induced subtree of $\mathcal{F}$. Namely, we can charge the points to the respective edges of $\mathcal{F}$, and the overall number of points in each level is bounded by $2(n-1)$.

The only type of points not counted above, is when `BuildNNTree` is called on a set with a single point (i.e., there is no edge to charge this point to). However, this case happens only when constructing an outer subtree. We can easily charge this case to the relevant parent, resulting in the $3n$ upper bound on the overall number of points stored in each level. ■

The search for an approximate NN for a query point q is performed by `FindApproxNN` as depicted in Figure 4.

Lemma 2.21 *The point returned by* `FindApproxNN`$(\mathcal{T}_P, q)$ *is a* 2γ*-NN to* q*, for* $1/2 \geq \gamma \geq 0$*, where* $\mathcal{T}_P =$`BuildNNTree`(P, γ).

Proof: Every time we descend one level in $\mathcal{T}$ through an outer edge, we lose a factor of $(1+\gamma/(3\log n))$ in the quality of approximation by Lemma 2.13. `FindApproxNN` stops when it resolves the NN-query by using $\mathcal{I}_v$, as $r_v^- \leq d_P(q) \leq r_v^+$. This introduces an error of at most a factor of $(1+\gamma/3)$.

Thus, the quality of approximation is

$$\begin{aligned}(1+\gamma/3)\left(1+\frac{\gamma}{3\log n}\right)^{\log n+1} &\leq (1+\gamma/3)\exp\left(\gamma(\log n+1)/(3\log n)\right)\\ &\leq (1+\gamma/3)\exp\left(\gamma/2\right) \leq 1+2\gamma.\end{aligned}$$

We conclude that the point returned is $(1+2\gamma)$-approximate NN to q in P. ∎

Lemma 2.22 *For $1/2 > \varepsilon > 0$, let $\mathcal{T}_P =$ `BuildNNTree`$(\mathcal{F}, P, \varepsilon/2)$, where $\mathcal{F}$ is a nd-stretch hierarchical clustering of P. Then $\mathcal{T}_P$ has the following properties:*

(i) Using `FindApproxNN` *on $\mathcal{T}_P$ answers ε-NN queries using $O(\log(n/\varepsilon))$* PLEB *queries.*

(ii) The overall number of PLEB*s used in $\mathcal{T}_P$ is $O\left(\frac{n}{\varepsilon}\log(n/\varepsilon)\right)$.*

(iii) The overall number of points *stored in the* PLEB*s of $\mathcal{T}_P$ is $O\left(\frac{n\log n}{\varepsilon}\log\frac{n}{\varepsilon}\right)$.*

(iv) One can implement `BuildNNTree` *in $O\left(\frac{n\log n}{\varepsilon}\log\frac{n}{\varepsilon}\right)$ time, ignoring the time required to construct the* PLEB*s themselves.*

Proof: **(i)** Consider the execution of `FindApproxNN` on $\mathcal{T}_P$, and let $\mathcal{I}$ be an interval PLEB encountered during the search. It has $K = O\left(\left(\log\frac{b}{a}\right)/\varepsilon\right)$ PLEBs, where $\frac{b}{a} \leq \frac{r_{top}(P)}{r_{median}(P)} = \frac{((36\lambda|P|\log n)/\varepsilon)r_{median}(P)}{r_{median}(P)} = O\left(\frac{n\log n}{\varepsilon}\right)$, as $\lambda = nd = O(n)$. Thus, $K = O\left(\frac{1}{\varepsilon}\log\frac{n\log n}{\varepsilon}\right) = O\left(\frac{\log(n/\varepsilon)}{\varepsilon}\right)$. If $r_v^- \leq d_P(q) \leq r_v^+$ then `FindApproxNN` performs a binary search on the internal PLEBs of $\mathcal{I}$. This requires $O(\log K)$ PLEB queries, but once this is done, the algorithm returns the approximate NN, and terminates the search. Otherwise, either $d_P(q) \leq r_v^-$ or $d_P(q) \geq r_v^+$. In both cases only two PLEB queries are being performed. Since the depth of the tree is $O(\log n)$, we conclude that the overall number of PLEB queries performed by the algorithm is $O(\log K + \log n) = O\left(\log\frac{\log(n/\varepsilon)}{\varepsilon} + \log n\right) = O\left(\log\frac{n}{\varepsilon}\right)$, as can be easily verified.

(ii) Arguing as in the proof of Lemma 2.20, we can interpret the construction of $\mathcal{T}_P$ as a recursive splitting of the edges of $\mathcal{F}$. It follows that the overall number of nodes of $\mathcal{T}_P$ is $O(n)$. Since each node of $\mathcal{T}_P$ have at most K PLEBs, it follows, that the overall number of PLEBs in $\mathcal{T}_P$ is $O\left(n\frac{\log(n/\varepsilon)}{\varepsilon}\right)$.

(iii) By Lemma 2.20, each level of $\mathcal{T}_P$ stored $\leq 3n$ points. Each such point might participate in K PLEBs, and the depth of $\mathcal{T}_P$ is $O(\log n)$.

(iv) We construct a *nd*-stretch clustering $\mathcal{F}$ of P in $O(n\log^2 n)$ time, using the algorithm of Lemma 2.9. This provides us with the r_{loss} value for each of the points of P. Next, implementing `BuildNNTree` in the time stated is straightforward, using the bound of (iii). Overall, the running time is $O\left(\left(\frac{1}{\varepsilon}\log\frac{n}{\varepsilon} + \log n\right)n\log n\right) = O\left(\frac{n\log n}{\varepsilon}\log\frac{n}{\varepsilon}\right)$. ∎

We can now strengthen our data-structure, by allowing approximate PLEB.

Definition 2.23 Let P be a set of points in $\mathbb{R}^d$, and $r > 0$, and $\gamma > 0$ parameters. An *γ-approximate point-location in equal ball data-structure*, denoted by $\gamma - \text{PLEB}(P, r)$, is a data-structure built on the points of P, so that given a query point $q \in \mathbb{R}^d$, it decides whether $q \in \mathbb{UB}_P(r)$. However, it is allowed to return a positive answer if $q \in \mathbb{UB}_P((1+\gamma)r)$. It *must* return a negative answer only if $q \notin \mathbb{UB}_P((1+\gamma)r)$. Furthermore, if it returns a positive answer, then it returns a point u of P such that $q \in B(u, (1+\gamma)r)$.

Theorem 2.24 *One can modify* `BuildNNTree`*, so that* $T = \texttt{BuildNNTree}(P, \varepsilon/18)$ *answers ε-NN queries using $O(\log(n/\varepsilon))$* PLEB *queries. Furthermore, this holds when* `BuildNNTree` *uses* $(\varepsilon/9)$-PLEB*s instead of exact* PLEB*s.*

Proof: The proof is similar to the proof of Lemma 3.6 below, and is thus omitted. ∎

Remark 2.25 An equivalent result to Theorem 2.24 was previously proved by Indyk and Motwani [IM98]. However, our reduction is considerably simpler, more efficient, and has other applications as demonstrated below.

3 NN Search via Point-location among Balls and Cubes

3.1 Point-Location among Balls

In this section, we consider the following problem: Given a set of points P, generate a small set $\mathcal{B}$ of balls so that for any query point q, an approximate NN to q can be computed by finding the ball in $\mathcal{B}$ that contains q.

Definition 3.1 For a set of balls $\mathcal{B}$ such that $\bigcup_{b \in \mathcal{B}} b = \mathbb{R}^d$ (i.e., one ball is infinite and covers the whole space), and a query point $q \in \mathbb{R}^d$, the *canonical ball of q in $\mathcal{B}$* is the smallest ball of $\mathcal{B}$ that contains q (if several equal radius balls contain q we resolve this in an arbitrary fashion).

Lemma 3.2 *For an interval* PLEB $\mathcal{I} = \mathcal{I}(P, a, b, \gamma)$*, one can compute a set $\mathcal{B}$ of $O((|P|/\gamma)\log(b/a))$ balls, so that for a query point q, if $a \le d_P(q) \le b$, then the canonical ball containing q in $\mathcal{B}$ corresponds to a point which is γ-NN to q in P.*

Proof: Observe, that each $PLEB(P, r)$ can be realized by a set of $|P|$ balls of radius r centered at the points of P. Let $\mathcal{B}$ be the set of balls which is the union of the set of balls generated from each PLEB of $\mathcal{I}$.

Since the canonical point-location gives preference to smaller balls, it follows that performing a point-location in $\mathcal{B}$ is equivalent to performing a binary search among the PLEBs of $\mathcal{I}$ and the lemma follows. ∎

At this stage, a natural solution to the problem mentioned above would be to take the tree generated by `BuildNNTree`, and transform all the IPLEBs stored in it into a set of balls, as described in Lemma 3.2. For a node $v \in \mathcal{T}_P$, let $\text{Balls}(v)$ denote the set of balls resulting for applying this process to the subtree of v. Namely, let $\text{Balls}(v) = \mathcal{B}(P_v, r_v^{max}) \cup$

$\text{Balls}(\mathcal{I}_v) \cup \bigcup_{u \text{ child of } v} \text{Balls}(u)$, where $\mathcal{B}(P_v, r_v^{max}) = \left\{ b(p, r_v^{max}) \;\middle|\; p \in P_v \right\}$. The following lemma is straightforward.

Lemma 3.3 *For a node $v \in \mathcal{T}_P$, and any two (inner) children v_X, v_Y, the sets of balls $\text{Balls}(v_X)$ and $\text{Balls}(v_Y)$ are disjoint. Furthermore, for any $b_X \in \text{Balls}(v_X)$, $b \in \text{Balls}(v_{\mathcal{I}})$, and $b_{outer} \in \text{Balls}(v_{outer})$, we have $\text{r}(b_X) \leq \text{r}(b) \leq \text{r}(b_{outer})$.*

Theorem 3.4 *Let P be a set of n points in $\mathbb{R}^d$. Given a parameter $\varepsilon > 0$, one can compute a set $\mathcal{B}(P)$ of $O\left((n/\varepsilon)(\log n)\log(n/\varepsilon)\right)$ balls, so that given any query point q, the center of the canonical ball $b \in \mathcal{B}$ that contains P, is an ε-NN to q in P.*

Proof: Let $\mathcal{T}_P$ be the NN-tree computed by `BuildNNTree`, so that $\mathcal{T}_P$ answer $\varepsilon/3$-NN queries. Let $\mathcal{B}(P) = \text{Balls}(\mathcal{T}_P)$. The bound on the size of $\mathcal{B}(P)$ follows from Lemma 2.22 (iii).

We claim that for a node v of height i a point-location query in $\text{Balls}(v)$ returns an $(1+\gamma)^i(1+\varepsilon/3)$ approximate NN to the query point q among the points of P_v, where $\gamma = \varepsilon/(3\log n)$.

If the height of v is 1 then claim trivially holds. Otherwise, consider the interval PLEB $\mathcal{I}_v$ of v. If $d_{P_v}(q) \geq r_v^+$, then all the balls in $\text{Balls}(v) \setminus \text{Balls}(v_{outer})$ do not contain q, and the search continues recursively into v_{outer}. Clearly, the quality of approximation is $(1+\gamma)^{h(v)}(1+\varepsilon/3)$ by induction.

If $d_{P_v}(q) \leq r_v^-$, then let $X \in \mathcal{CC}(P_v, r_v^-)$ be the set of $\mathcal{CC}_{P_v}(r_v^-)$ that contains q. By induction, when we perform a point-location query in $\text{Balls}(v_X)$, we find a ball $b \in \text{Balls}(v_X)$ which contains q, and its center is an $(1+\gamma)^{h(v_X)}(1+\varepsilon/3)$ approximation to the NN to q in X. We observe that all the balls in $\text{Balls}(v) \setminus \text{Balls}(v)$ are either have radius larger than r_v^-, or alternative, are induced by points belonging to other sets of $\mathcal{CC}_{P_v}(r_v^-)$. In later case, those balls can not contain q. Thus, performing a point-location query in $\text{Balls}(v_X)$ with q, is equivalent to performing the query in $\text{Balls}(v)$ with q.

If $r_v^- < d_{P_v}(q) \leq r_v^+$ then the only set of balls of $\text{Balls}(v)$ which are relevant are $\text{Balls}(\mathcal{I}_v)$. Clearly, by the definition of the interval PLEB $\mathcal{I}_v$, the canonical ball of q in $\text{Balls}(\mathcal{I}_v)$ is defined by an $\varepsilon/3$-NN point to q among the points of P_v. ∎

Definition 3.5 For a ball $b = b(p, r)$ centered at p of radius r, a set C is an ε-approximation to b, if $b \subseteq C \subseteq b(p, (1+\varepsilon)r)$. For a set of balls $\mathcal{B}$, the set $\mathcal{C}$ is an ε-approximation to $\mathcal{B}$ if for any ball $b \in \mathcal{B}$ there is a corresponding ε-approximation $C_b \in \mathcal{C}$. For a set $C \in \mathcal{C}$, let $b(C) \in \mathcal{B}$ denote the ball corresponding to C, and let $p(C)$ denote the center of $b(C)$.

Given a point q, the *canonical set of $\mathcal{C}$ that contains q* is the set of $\mathcal{C}$ that contains q and is associated with the smallest radius ball of $\mathcal{B}$.

Lemma 3.6 *Let $\mathcal{T}_P$ be the tree returned by `BuildNNTree`$(P, \varepsilon/6)$, $\mathcal{B} = \text{Balls}(\mathcal{T}_P)$, and $\mathcal{C}$ an $\varepsilon/9$ approximation to $\mathcal{B}$. Let q be an arbitrary query point in $\mathbb{R}^d$, and C be the canonical set of $\mathcal{C}$ containing q. Then $p(C) \in P$ is an ε-NN to q in P.*

Proof: We claim that if a node v has height i, P_v contains a $(1+\alpha)$-NN to q, and $r_v^{min} \leq d_P(q) \leq r_v^{max}$, then a point-location query in $\mathcal{C}(\text{Balls}(v))$ returns an $(1+\varepsilon/3)(1+\gamma)^i(1+\alpha)$ approximate NN to the query point q among the points of P_v, where $\gamma = \varepsilon/(3\log n)$.

If the height of v is 1 then claim trivially holds, and we omit the easy proof. Otherwise, one of the following holds:

- $d_{P_v}(q) \leq r_v^-(1-\varepsilon/9)$: Let $X \in \mathcal{CC}(P_v, r_v^-)$ be the set of $\mathcal{CC}_{P_v}(r_v^-)$ that contains q. We observe that all the sets in $\mathcal{C}(v) \setminus \mathcal{C}(v_X)$, either have radius larger than r_v^-, or alternatively, are induced by points belonging to other sets of $\mathcal{CC}_{P_v}(r_v^-)$. In the later case, those sets can not contain q, as their distance from q is at least $(1+\varepsilon/9)r_v^-$, and the radius of the balls used to define them is $\leq r_v^-$. Thus, performing a point-location query in $\mathcal{C}(v_X)$ with q, results in the same set as performing the query in $\mathcal{C}(v)$ with q. By induction, when we perform a point-location query in $\mathcal{C}(v_X)$, we find a set $C \in \mathcal{C}(v_X)$ which contains q, and its center is an $(1+\varepsilon/3)(1+\gamma)^{i-1}(1+\alpha)$ approximation to the NN to q in X.

- $(1-\varepsilon/9)r_v^- \leq d_{P_v}(q) \leq r_v^-(1+\varepsilon/9)$: If $C \in \mathcal{C}(v) \setminus (\mathcal{C}(\mathcal{I}_v) \cup \mathcal{C}(v_{outer}))$, then this set corresponds to a point $p = p(C)$ of distance $\leq (1+\varepsilon/9)r_v^-$ from q. Namely, $\|pq\| \leq (1+\varepsilon/9)r_v^- \leq (1+\varepsilon/9)\frac{d_{P_v}(q)}{1-\varepsilon/9} \leq (1+\varepsilon/3)d_{P_v}(q)$.

  Otherwise, it must be that $r_v^- \leq d_{P_v}(q) \leq r_v^-(1+\varepsilon/9)$, and q is contained inside a set $C \in \mathcal{C}(\mathcal{I}_v)$. The distance of q to $p(C)$ is at most $(1+\varepsilon/9)(1+\varepsilon/(3\cdot 6))d_{P_v}(q)$, and the claim holds.

- $r_v^-(1+\varepsilon/9) \leq d_{P_v}(q) \leq r_v^+$: Only the sets in $\mathcal{C}(\mathcal{I}_v)$ are relevant, and the point-location resolve the query correctly, with error at most $(1+\varepsilon/9)(1+\varepsilon/18)d_{P_v}(q)$.

- $r_v^+ \leq d_P(q) \leq (1+\varepsilon/9)d_P(q)$: If q is contained inside a set C of $\mathcal{C}(v)\setminus\mathcal{C}(v_{outer})$, then we are done, as $p(C)$ is an $\varepsilon/3$-NN to q as can be easily verified. Otherwise, performing a point-location query in $\mathcal{C}(v)$ is equivalent to performing a point-location query in $\mathcal{C}(v_{outer})$, which implies the correctness of the result by induction, as v_{outer} contains an $(1+\gamma)(1+\alpha)$-NN to q, by Lemma 2.13, and its height is $\leq i-1$.

- $(1+\varepsilon/9)r_v^+ \leq d_{P_v}(q)$: All the sets in $\mathcal{C}(v)\setminus\mathcal{C}(v_{outer})$ do not contain q, and the search continues recursively into v_{outer}. The claim now follows by induction and by Lemma 2.13.

Applying the claim to the root of $\mathcal{T}_P$ with $\alpha = 0$, it follows that the quality of NN found is $(1+\varepsilon/3)(1+\gamma)^{\lceil \log n+1\rceil}(1+0) \leq 1+\varepsilon$, as required. ∎

3.2 Point-location among cubes

For a real number u, let $\mathfrak{G}(u)$ be the partition of $\mathbb{R}^d$ into a uniform axis-parallel grid centered at the origin, where the side-length of the grid is $2^{\lfloor \log u\rfloor}$ (we define $\lfloor u\rfloor$ to be the largest integer number smaller than u). Note, that by varying the value of u, we get a multi-resolution grid that covers space.

For a ball $b = b(p,r)$, let $GC(b,\varepsilon)$ be the set of cubes of the grid $\mathfrak{G}(r\varepsilon/(3d))$ that intersects b. Let $b_\varepsilon = \cup_{c\in GC(b,\varepsilon)} c$. Clearly, b_ε is an ε-approximation to b. Note that $|GC(b,\varepsilon)| = O(1/\varepsilon^d)$. For a set of balls $\mathcal{B}$, let $GC(\mathcal{B},\varepsilon) = \cup_{b\in\mathcal{B}} GC(b,\varepsilon)$. For $c \in GC(\mathcal{B},\varepsilon)$, let $r(c)$ be the radius of the smallest ball $b \in \mathcal{B}$, such that $c \in GC(b,\varepsilon)$. Similarly, let $p(c)$ be the center of the ball that realizes $r(c)$.

Theorem 3.7 *Let P be a set of n points in $\mathbb{R}^d$. Given a parameter $\varepsilon > 0$, one can compute a set $\mathcal{C}(P)$ of $O\left(n\frac{\log n}{\varepsilon^d}\log\frac{n}{\varepsilon}\right)$ cubes, all of them taken from the hierarchical grid $\mathfrak{G}(\cdot)$. For*

any query point q, let $c(q)$ be the cube of C that contains it and has the smallest value of $r(c)$ associated with it. Then, $p(c)$ is an ε-NN to q in P. The running time needed to compute this set of cubes is $O\left(n\frac{\log n}{\varepsilon^d}\log\frac{n}{\varepsilon}\right)$.

Proof: Let $\mathcal{T}_P$ be the tree returned by `BuildNNTree` for $\varepsilon/3$, and let $\mathcal{B} = \text{Balls}(\mathcal{T}_P)$ be the associated set of balls. Let $\mathcal{C} = \cup_{b\in\mathcal{B}} GC(b, \varepsilon/9)$. Clearly, $\mathcal{C}$ is the required set of cubes. The correctness of the result follows from the observations that b_ε is an ε-approximation to b, for any $b \in \mathcal{B}$, and as such the result is correct by Lemma 3.6.

A naive bound on the size of $\mathcal{C}$ is $O\left(n\frac{\log n}{\varepsilon^{d+1}}\log\frac{n}{\varepsilon}\right)$. However, one can do better, as follows: For a point $p \in P$, and an interval PLEB $\mathcal{I} \in \mathcal{T}_P$, such that $p \in \mathcal{I}$, let $\mathcal{B}_{p,\mathcal{I}}$ be the set of balls in $\mathcal{B}$ centered at p that were created during the realization of $\mathcal{I}$ by balls. Clearly $|\mathcal{B}_{p,\mathcal{I}}| = O((\log n/\varepsilon)/\log(1+\varepsilon)) = O(\log(n/\varepsilon)/\varepsilon)$. However, those balls have a common center, and the smaller balls have higher priority in the point-location. Thus, we can throw out the cubes having low-priority covered by higher priority cubes. It follows, using a standard exponential grid argument, that the number of cubes in $C_{p,\mathcal{I}} = \cup_{b\in\mathcal{B}_{p,\mathcal{I}}} GC(b, \varepsilon/9)$ is only $O((\log n/\varepsilon)/(\varepsilon^d \log(2))) = O((\log n/\varepsilon)/\varepsilon^d)$. Since there are $O(n\log n)$ pairs $(p, \mathcal{I})$ of point/IPLEB in $\mathcal{T}_P$, It follows that the overall number of cubes in $\mathcal{C}$ is $O\left(n\frac{\log n}{\varepsilon^d}\log\frac{n}{\varepsilon}\right)$. ∎

Remark 3.8 In the set of cubes generated by Theorem 3.7 there is one infinite cube, that covers the whole space. This is a "background" cube, that serves for answering ε-NN queries when the query point is so far from P that any point of $x \in P$ is an ε-NN to the query point.

4 Algorithm and Applications

Consider the set of cubes computed by the algorithm of Theorem 3.7. Since all those cubes are taken from a hierarchical grid representation of $\mathbb{R}^d$, we can store all those cubes in a quadtree, and use the quadtree to answer point-location queries among those prioritized cubes (note that larger cubes, stored in higher levels of the quadtree have lower priority, as the radius of the ball that created them is larger), and thus approximate nearest-neighbor queries. By using compressed quadtrees, we have our main result:

Theorem 4.1 *Let P be a set of n points in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$, then one can compute a set $\mathcal{C}(P)$ of $O\left(n\frac{\log n}{\varepsilon^d}\log\frac{n}{\varepsilon}\right)$ regions. Each region is either a cube or an annulus (i.e., the difference between two cubes, one contained inside the other). The regions of $\mathcal{C}(P)$ are disjoint and provide a covering of space.*

Furthermore, each such region $c \in C$ has an associated point $p \in P$, so that for any point $q \in c$, the point p is a ε-NN of q in P. Thus, given a query point $q \in \mathbb{R}^d$, one can compute in $O(\log(n/\varepsilon))$ time a ε-NN to q in P.

The overall time to perform this construction is $O\left(n\frac{\log n}{\varepsilon^d}\log^2\frac{n}{\varepsilon}\right)$.

Proof: We use Theorem 3.7 to compute the required set $\mathcal{C}$ of cubes. Next, we store this set of cubes in a compressed quadtree. Using standard techniques [AMN+98], this compressed quadtree $\mathcal{Q}$ can be computed in $O(|\mathcal{C}|\log|\mathcal{C}|) = O\left(n\frac{\log n}{\varepsilon^d}\log^2\frac{n}{\varepsilon}\right)$ time. Finally, we need to dump all the leafs of $\mathcal{Q}$ (they provide the required covering of space). Note that since this is a compressed quadtree, a leaf either corresponds to a cube, or to the region between two cubes.

Finally, we note that we can preprocess the leafs of $\mathcal{Q}$ for point-location using the data-structure of [Fre85]. A point-location query, for a point q, can be answered in $O(\log |\mathcal{Q}|) = O(\log(n/\varepsilon))$ time. Since the point p_c associated with the cell found contains q is an ε-NN for all the points of c, it follows that we can answer ε-NN queries in $O(\log(n/\varepsilon))$ time. ∎

Remark 4.2 Theorem 4.1 provides an ε-approximation to a Voronoi diagram. The standard Voronoi diagram has complexity $O(n^{\lceil d/2 \rceil})$ in the worst case in d-dimensions. The dissection of space provided above is only approximate but has considerably smaller complexity. In particular, our solution has near-linear dependency on n.

Remark 4.3 The result of Theorem 4.1 provides the fastest currently known algorithm for answering approximate NN queries, with near-linear space. To our knowledge, all other approximate NN data-structures have query time with polynomial dependency on $1/\varepsilon$.

4.1 A point-set with bounded spread

The *spread* $\rho(P)$ of a point-set P is the ratio $\Delta(P)/\delta(P)$, where $\Delta(P)$ is the length of the diameter of P, and $\delta(P)$ is the distance between the closest pair of points of P. If one allows the algorithms to depend on $\rho(P)$, then one can generate considerably simpler data-structures with (potentially) better performance. The results above are interesting in the cases where $\rho(P)$ is very large (i.e., exponential in n); namely, the input is highly clustered.

Lemma 4.4 *Given a set P of n points in $\mathbb{R}^d$ and a bounding cube U that contains it, one can compute a set $\mathcal{C}$ of $O((n/\varepsilon^d)\log(\rho(P)/\varepsilon))$ disjoint cubes that covers U, such that for each cube $c \in \mathcal{C}$ there is a point $p_c \in P$ associated with it, so that p_c is a ε-NN for all points inside c. Furthermore, one can answer approximate NN queries inside U by finding the cube of $\mathcal{C}$ that contains the query point.*

Proof: We observe, that to resolve an approximate NN query in such setting, it is enough to construct a single interval PLEB $\mathcal{I} = \mathcal{I}(P, \delta(P)/2, 3\Delta(P)/\varepsilon, \gamma)$. Arguing as in the proof of Theorem 3.7, we generate the corresponding set of cubes and the result follows. ∎

Remark 4.5 The approximate NN problem is easy when $\rho(P)$ is small. For relevant results see [AEIS99, Car01] on fast data-structures for answering approximate NN queries. Similar results to [AEIS99] follows by using stratified trees [vE77] with Lemma 4.4.

A *α-fat decomposition* $\mathcal{D}$ of a cube U, is a disjoint covering of C by simplices, so that if two simples are adjacent, than their diameters are the same up to a factor of α, and furthermore, for any $\sigma \in \mathcal{D}$ the ratio between the smallest ball containing σ, and the largest ball contained in σ is bounded by α.

It is well known, that given a quadtree $\mathcal{Q}$ of depth L with n-nodes then one can compute a set of $O(nL)$ cubes that cover space, which provides a refinement to the decomposition of space provided by $\mathcal{Q}$ and has a constant ratio between the sizes of two cubes that share a common facet [BEG94]. Clearly, by triangulating any of those cubes, the resulting decomposition is going to be α-fat, for an appropriate constant α that depends only on the dimension. Note, that the smallest ball generated by the algorithm constructing the set of balls $\mathrm{Balls}(P, \varepsilon)$ is of radius $O(\varepsilon\delta(P)/\log n)$. Putting everything together, we conclude:

Theorem 4.6 *Let P be a set of n points in $\mathbb{R}^d$, and let U be a cube that contains P such that $\mathrm{diam}(U) \leq (\mathrm{diam}(P))^2/\varepsilon$. Given a parameter $\varepsilon > 0$, one can compute a α-fat decomposition of U into $O\left(\frac{n}{\varepsilon^d} \log \frac{\rho(P)}{\varepsilon}\right)$ simplices, where α is an appropriate constant independent of ε. Furthermore, each simplex $c \in C$ has an associated point $p \in P$, so that for any c point $q \in c$, the point p is a ε-NN of q in P.*

In the plane, this result can be turned into fat triangulation (i.e., the above decomposition is not a valid simplicial complex) by using the techniques of Bern *et al.* [BEG94].

Theorem 4.7 *Let P be a set of n points in the plane, ε a parameter larger than zero, and U a square that contains P such that $\mathrm{diam}(U) \leq (\mathrm{diam}(P))^2/\varepsilon$. Then, one can compute a α-fat triangulation T of U made out of $O\left(\frac{n}{\varepsilon^2} \log \frac{\rho(P)}{\varepsilon}\right)$ triangles, where α is an appropriate constant independent of ε. Furthermore, each triangle $c \in T$ has an associated point $p \in P$, so that for any c point $q \in c$, the point p is a ε-NN of q in P. This triangulation can be computed in $O\left(\frac{n}{\varepsilon^2} \log^2 \frac{\rho(P)}{\varepsilon}\right)$ time.*

5 Conclusions

In this paper, we presented (to our knowledge) the first decomposition of space that approximates a Voronoi diagram and has near linear size. The updated version of this paper is available online [Har01].

There are numerous open questions for further research:

- Can the construction be improved and yield a smaller decomposition?
- The construction is currently indirect, going through the usage of PLEBs. It would be nice to avoid this and come up with a direct and simpler construction.
- Can one maintain such a decomposition of space efficiently for moving points?
- Can one come up with a better construction of Interval PLEB for constant dimension?
- Can one define a similar notion of approximate Voronoi diagram for a set of lines in 3D. It is clear that quadratic complexity is a lower bound, by the ruled surface construction of Chazelle [Cha84]. However, even a quadratic bound would be interesting in this case, as the bound for the exact Voronoi diagram is still open. The question is especially interesting for the case of Voronoi diagram defined by a surface in 3D. Such a construction would be useful for algorithms doing shape simplification.

Acknowledgments

The author wishes to thank Pankaj K. Agarwal, Alon Efrat, Jeff Erickson, Piotr Indyk and Edgar Ramos for helpful discussions concerning the problems studied in this paper. The author also wishes to thank David Bunde for his comments on the write-up.

References

[AB01] D. Attali and J.D. Boissonnat. Complexity of the delaunay triangulation of points on a smooth surface. http://www-sop.inria.fr/prisme/personnel/boissonnat/papers.html, 2001.

[AEIS99] A. Amir, A. Efrat, P. Indyk, and H. Samet. Efficient algorithms and regular data structures for dilation, location and proximity problems. In *Proc. 40th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 160–170, 1999.

[AMN+98] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. Assoc. Comput. Mach.*, 45(6), 1998.

[AS99] P.K. Agarwal and M. Sharir. Pipes, cigars, and kreplach: The union of Minkowski sums in three dimensions. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 143–153, 1999.

[Aur91] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, 1991.

[BEG94] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *J. Comput. Syst. Sci.*, 48:384–409, 1994.

[Car01] M. Cary. Towards optimal ϵ-approximate nearest neighbor algorithms in constant dimnesions. from web page, 2001.

[Cha84] B. Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM J. Comput.*, 13:488–507, 1984.

[Cha98] T.M. Chan. Approximate nearest neighbor queries revisited. *Discrete Comput. Geom.*, 20:359–373, 1998.

[CK95] P.B. Callahan and S.R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42:67–90, 1995.

[Cla94] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 160–164, 1994.

[DHS01] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification.* Wiley-Interscience, New York, 2nd edition, 2001.

[Eri01] J. Erickson. Nice points sets can have nasty delaunay triangulations. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, 2001. To appear. Available from http://compgeom.cs.uiuc.edu/~jeffe/pubs/spread.html.

[Fre85] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985.

[Har01] S. Har-Peled. A replacement for voronoi diagrams of near linear size. manuscript. Available from `http://www.uiuc.edu/~sariel/papers`, 2001.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 604–613, 1998.

[KOR98] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 614–623, 1998.

[vE77] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inform. Process. Lett.*, 6:80–82, 1977.