Available online at www.sciencedirect.com
ScienceDirect
Journal of Computer and System Sciences 72 (2006) 955–977
JOURNAL OF COMPUTER AND SYSTEM SCIENCES
www.elsevier.com/locate/jcss

Nearest neighbors search using point location in balls with applications to approximate Voronoi decompositions ☆

Yogish Sabharwal [a], Nishant Sharma [b], Sandeep Sen [b,*]

[a] *IBM India Research Lab, Block-I, IIT Delhi, Hauz Khas, New Delhi 110016, India*
[b] *Department of Computer Science and Engineering, IIT, New Delhi 110016, India*

Received 13 March 2003; received in revised form 27 December 2005

Available online 7 July 2006

Abstract

We present alternate reductions of the nearest neighbor searching problem to Point Location in Balls that reduces the space bound of Sariel Har-Peled's [S. Har-Peled, A replacement for Voronoi diagrams of near linear size, in: Proc. of IEEE FOCS, 2001, pp. 94–103, full version available from http://www.uiuc.edu/~sariel/papers] recent result on Approximate Voronoi Diagrams to linear while maintaining the logarithmic search time. We do this by simplifying the construction of [S. Har-Peled, A replacement for Voronoi diagrams of near linear size, in: Proc. of IEEE FOCS, 2001, pp. 94–103, full version available from http://www.uiuc.edu/~sariel/papers] that reduces the number of balls generated by algorithm by a logarithmic factor to $O(n \log n)$. We further reduce the number of balls by a new hierarchical decomposition scheme and a generalization of PLEBs to achieve linear space decomposition for nearest neighbor searching. The construction of our data structures takes $O(n \log n)$ time.

Keywords: Voronoi diagrams; Approximate nearest neighbor; Data structures

1. Introduction

Let P be a set of n points in a metric space. One of the most fundamental problems with diverse applications is to build a data structure on P that supports nearest neighbor searching efficiently. This problem has satisfactory solutions when P is planar but becomes non-trivial in higher-dimensional Euclidean space even in three dimensions [5,9]. The most general approach to nearest neighbor searching is to build a Voronoi diagram of P (to be denoted as $\mathrm{Vor}(P)$) which is a partition of the space into cells such that a cell consists of all points that are closest to a specific point of P than any other point of P. Voronoi diagrams are fundamental computing tools in computational geometry that have many other applications, including clustering, motion planning, learning, surface reconstructions etc. Generalization of Voronoi diagrams to objects beyond point sets are also equally useful (refer to Aurenhammer [6] for a detailed survey). Despite its numerous applications, Voronoi diagrams suffer from the drawback of high structural complex-

☆ A preliminary version of this paper appeared in the Proceedings of the 22nd Annual FSTTCS 2002, Kanpur, India, Lecture Notes in Comput. Sci., vol. 2556, Springer-Verlag.

* Corresponding author.
E-mail addresses: ysabharwal@in.ibm.com (Y. Sabharwal), nishantt@cse.iitd.ernet.in (N. Sharma), ssen@cse.iitd.ernet.in (S. Sen).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2006.01.007

ity (and consequent computational bottleneck). It is known that the worst case complexity in $\mathbb{R}^d$ is $\theta(n^{\lceil d/2 \rceil})$ with constants exponential in d. Therefore building and storing these diagrams becomes computationally infeasible with growing dimensions.

As a result, alternate solutions for nearest neighbor searching in higher dimensions have been a hot topic of research in recent years [1,4,8,12–14]. Since the exact problem seems intimately related to the complexity of Voronoi diagrams, the related relaxed problem of *approximate nearest neighbor search* has gained importance and has also shown promise of practical solutions. Given a constant $\varepsilon < 1$, we want to preprocess the data-set P to get a data structure D, such that given a query point q, we can efficiently return a $p \in P$ such that $\forall p' \in P, \text{dist}(p, q) \leqslant (1 + \varepsilon)\,\text{dist}(p', q)$. Here, dist could be any arbitrary metric, and then p is an ε nearest neighbor (ε-NN) of q.

In the context of this paper, the work of Indyk and Motwani [12] holds special significance where they reduced the problem of ε-NNS to Approximate Point Location in Equal Balls (ε-PLEB).

Definition 1.1. Given P and a parameter r, an ε-PLEB(P, r) is a data structure which when given a query point q returns a point $p \in P$ such that $\text{dist}(p, q) \leqslant r$, if there exists one. If there is no point $p' \in P$ with $\text{dist}(p', q) \leqslant r(1+\varepsilon)$, then it returns *nil*. Otherwise, it can return anything. When $\varepsilon = 0$, then it is called PLEB(P, r).

Indyk and Motwani used a novel ring-cover tree to reduce ε-NNS to ε-PLEB. The ε-PLEB is solved by using a simple hash-based technique. However, their reduction was quite complex which was improved in all respects in the work of Har-Peled [11] who presented an alternate decomposition of space called an *Approximate Voronoi Diagram* (to be referred as *AVD*) that supports approximate nearest neighbor searching and has a significantly lower space complexity. His result can be summarized as follows.

Theorem 1.1 *(Har-Peled). Given P of n points in $\mathbb{R}^d$ and a parameter $\varepsilon > 0$ one can compute a set $\mathcal{C}(P)$ of $O(n\frac{\log n}{\varepsilon^d}\log n/\varepsilon)$ regions where each region is a cube or an annulus of cubes. The regions of $\mathcal{C}(P)$ are disjoint and cover the space and each region 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 and it can be computed in $O(\log(n/\varepsilon))$ steps.*

The time for constructing the data structure is $O(n\frac{\log n}{\varepsilon^d}\log^2 n/\varepsilon)$.

Har-Peled also used a more sophisticated data structure for solving ε-NNS based on the BBD data structure of Arya et al. [1].

1.1. Our results

An important question left open in Har-Peled's work was if *AVD* could be made linear-sized rather than *near linear*. In this paper, we achieve reduction in space by extending some of the ideas of Har-Peled and generalizing the notion of PLEBs to *Point Location in Smallest Ball* (PLSB) where the balls may be of different radii. We also define its approximate version—for precise definitions, the reader may refer to Section 4. Har Peled [11] had also addressed this problem implicitly—in a more ad-hoc manner.

Note that PLEB is a special case of PLSB, with all balls of equal radius.

Lemma 1.1. *An ε-PLSB on n balls can be solved for d-dimensional space endowed with metric l_p, in $O(\log n)$ query time and $O(n/\varepsilon^d)$ space.*

The method is similar to Har-Peled [11]—we have included a summary in Appendix C. *In the remaining paper we will only bound the number of balls in the data structure which is linearly related to the space complexity.*

In the next section, we eliminate the recursive structure of the Har-Peled's algorithm by constructing a set of balls directly from the clustering and also reduce the number of balls by a factor of $\log n$. The search time remains logarithmic and the preprocessing time is also improved by a factor of $O(\log n)$.

In Section 4, we improve the space bound to linear using a new hierarchical clustering scheme that is also based on the MST of P. To reduce the preprocessing time (the exact MST construction takes nearly quadratic time), we use a faster approximate MST algorithm that takes $O(n \log n)$ time—the same as Har-Peled. In Har-Peled's data structure we always merged two clusters, if the MST edge connecting them is smallest. The difference between hierarchical

clustering introduced here and that of Har-Peled's is—here instead of trying to merge two nearest clusters, we merge a number of the clusters in phases such that diameter of the new cluster generated is small (Section 4).

In an independent work, Arya and Malamatos [2] have also achieved a similar result based on the well separated space decomposition of Callahan and Kosaraju [7]. Further, they obtain space-time trade-offs for generalization of the Voronoi cells with more than one representative—this has been further improved by Arya at al. [3]. One possible advantage of our method is that the number of balls generated in our algorithm is much less—$O(n)$ as compared to $O((\alpha\sqrt{d})^d n)$ and an alternate search structure (other than the BBD tree) may be able to exploit this.

2. A brief review of Har Peled's construction

Har-Peled [11], proposes a solution based on the following observation:

Observation 2.1. *Let the distance between two points A and B be $|AB|$. If a query point Q lies further than $k \cdot |AB|$ from A and B, where k is set to $1/\varepsilon$, then both A and B are ε-neighbors of Q, i.e., $\frac{|QA|}{|QB|} < 1 + \epsilon$.*

Har-Peled [11] further proposes a clustering method for clustering the points into a hierarchical tree. Consider the connected components obtained by growing balls around all the points. We also construct a forest (called cluster tree) from the connected component.

Initially we start out with just the set of points themselves. Thus we start with n connected components consisting of the individual points. The corresponding forest consists of n trees with each tree consisting of a single leaf, corresponding to a point.

We then grow the balls around all the points uniformly. When two components meet, we get a new component that replaces the two components corresponding to the points whose balls meet and is a result of merging the two components. In the corresponding forest, we merge the two trees corresponding to the components being merged by hanging one of the trees on the other one arbitrarily (see Fig. 1).

At this point, we also associate a value of $r_{\text{loss}}(p)$ with the root, p of the tree that we hang on the other one. This value corresponds to the radius of the balls when these components meet. It is actually half the distance between the closest points of the two components.

Thus $r_{\text{loss}}(p) =$ radius of the ball at p, when p ceases to be root.

2.1. Properties of the cluster tree

- Values of r_{loss} increase along a path towards the root.
- If $L_{\max}$ is the longest edge of the MST, the tree edge corresponding to this is the last edge to be added.

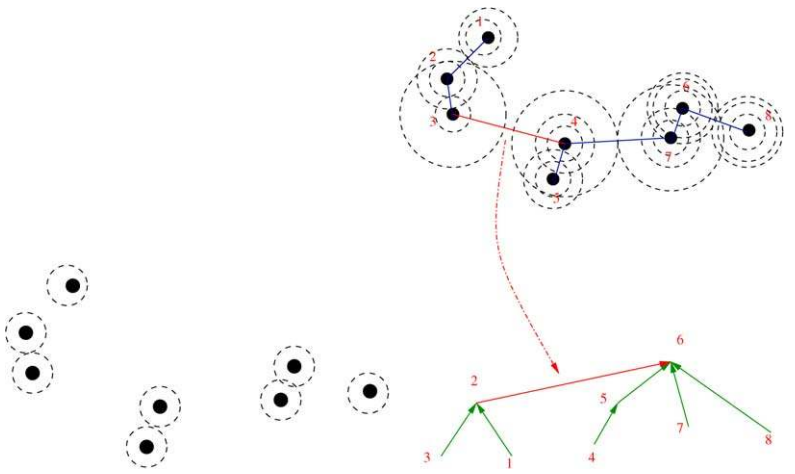

Fig. 1. Constructing the cluster tree.

- If any query point q that is outside of the union of balls of radii $L_{\max}.|P|.1/\epsilon$ centered at $p \in P$, then any point p is an ϵ-nearest neighbor of q, i.e., q is too far from P. Note that the diameter of P is bounded by $L_{\max}.|P|$.

The above properties also hold for any subtree (cluster) formed during the construction.

Har-Peled, instead of working with an exact MST in d-dimensions which takes $O(n^2)$ time to compute for large d, settles for an nd factor approximation that can be computed in $O(n \log n)$ time.

Using the nd-MST, he constructs an nd-stretch Hierarchical Clustering and then gives an algorithm for the construction of a family of PLEBs (Point Location in Equal Balls) for answering the approximate nearest neighbor queries in time $O(\log(\frac{n}{\epsilon}))$. The search proceeds by pruning some of the clusters on the basis of some PLEBs and recursively searching the remaining points. For each recursive call we have to construct $O(\log n)$ PLEBs and there are $O(\log n)$ levels. Finally the $O(n \log^2 n)$ balls are merged into a single search data structure (a compressed quadtree [1]). The bound on the space depends on the number of balls generated by the family of PLEBs which is roughly $O(n \log^2 n)$.

3. Improving the space bound

In this section we present a different perspective, whereby we decrease the number of balls required for answering approximate nearest neighbor queries by pruning overlapping and unwanted balls from the set of balls centered around a point $p \in P$. We give an independent algorithm for the construction of balls around the points of P, such that reporting the center of the smallest ball that contains a query point answers the approximate nearest neighbor query.

3.1. Building the nd-hierarchical clustering

The following definitions are similar to Har-Peled that have been restated for convenience of the reader.

Definition 3.1 *(λ-MST).* A tree T having the points of P in its nodes, is a λ-MST of P, if it is a spanning tree of P, having a value len(.) associated with each edge of T, such that for any edge $e = uv$ of T, $\text{len}(e) \leqslant d(P_1, P_2) \leqslant \lambda\,\text{len}(e)$, where P_1, P_2 is the partition of P into two sets as induced by the two connected components of T-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.

One can compute a nd-MST of P in $O(n \log n)$ time. Har-Peled [11] shows how such an nd-MST can be constructed, and is similar to the fair-split tree construction of Callahan and Kosaraju [7].

Definition 3.2 *(λ-Stretch hierarchical clustering).* A Directed tree T storing the points of P in its nodes, so that for any point p, there is a value $r_{\text{loss}}(p)$ associated with the out-edge emanating from p to its parent, satisfying the following property:

If C_1 denotes connected components obtained on constructing balls of radius r around all points, C_2 denotes connected components obtained on constructing balls of radius λr around all points, F is the forest obtained by removing from T, all edges larger than r and X is the partition of P into subsets induced by the connected components of the forest F, then $C_1 \leqslant X \leqslant C_2$, where $X \leqslant Y \Rightarrow$ if A and B are in the same component in X, then they are in the same component in Y.

The nd-stretch hierarchical clustering is built from the nd-MST as follows:

We sort the edges of the nd-MST in order of the len(.) values of the edges. We then traverse the list. In each iteration, we take the next edge, say $e = xy$, from the sorted list and connect the two subtrees to which x and y belong by hanging the smaller tree onto the larger one by introducing an edge between the roots of the two subtrees. We associate with this edge e, the value $r_{\text{loss}}(e) = \text{len}(xy)/2$.

For the root, t, of the tree, we assign $r_{\text{loss}}(t) = \max(r_{\text{loss}}(p_i)$, where p_i is a child of $t)$.

Observation 3.1. *The height of the tree formed in the λ-stretch hierarchical clustering of P is at most* $\log n$. (*Hanging the smaller tree below the larger one insures this property.*)

We will use $b(p, r)$ to denote the ball of radius r centered at p.

Definition 3.3 *(r_{death}, r_{low}, r_{high}).* For an approximation factor γ, define $r_{\text{death}}(p) = 6\lambda r_{\text{loss}}(p)n \log n/\gamma$ and define $r_{\text{low}}(p) = (1/(1+\gamma/3))r_{\text{loss}}(p)$. $r_{\text{low}}(p)$ gives us a value just smaller than $r_{\text{loss}}(p)$, s.t., if a query point $q \notin b(p, r_{\text{low}}(p))$ but $q \in b(p, r_{\text{loss}}(p))$, then p is a $(1+\gamma/3)$-approximate NN of the query point. Also, define $r_{\text{high}}(p) = (36\lambda r_{\text{loss}}(p)n \log n/\gamma)$. Note that for any point p, $r_{\text{high}}(p) > r_{\text{death}}(p)$.

Definition 3.4 *(parent(p), parent$^i(p)$).* Define parent(p) to be the parent of node p in the tree obtained by the λ-stretch hierarchical clustering of P. Define parent$^i(p)$ to be the ith parent of node p in the tree obtained by the λ-stretch hierarchical clustering of P, i.e., parent$^0(p) = p$ and parent$^i(p) =$ parent(parent$^{(i-1)}(p)$). Note that parent$^i(p)$ is defined till it is the root of the tree obtained by the λ-stretch hierarchical clustering of P, and is not defined beyond the root.

We will use Subtree(p) to denote the subtree rooted at p.

3.2. Construction of balls (algorithm ConstructBalls(P, γ))

In this section $\lambda = nd$. Given a λ-stretch hierarchical clustering of P, for each point p in P, let r_0 be the r_{loss} value for p and let $p_1, p_2, \ldots, p_m$ be the children of p in sorted order of their r_{loss} values, i.e., $r_{\text{loss}}(p_1) \leqslant r_{\text{loss}}(p_2) \leqslant \cdots \leqslant r_{\text{loss}}(p_m)$. Also, let x be the parent of p in the tree formed by the λ-stretch hierarchical clustering of P.

We construct the following ball sets around p:

(1) Balls with radius $r_i = r_{\text{loss}}(p)(1+\gamma/3)^{(j-1)}$ for $j = 0, \ldots, M-1$, where $M = \lceil \log_{(1+\gamma/3)}(1+\gamma/3)(r_{\text{high}}(p)/r_{\text{loss}}(p))\rceil$.

This defines a ball set in the range $[r_{\text{low}}(p), r_{\text{high}}(p)]$.

(2) Balls with radius $r_i = r_{\text{loss}}(p_i)(1+\gamma/3)^{(j-1)}$ for $j = 0, \ldots, M-1$, where $M = \lceil \log_{(1+\gamma/3)}(1+\gamma/3)(r_{\text{high}}(p_i)/r_{\text{loss}}(p_i))\rceil$.

This defines a ball set in the range $[r_{\text{low}}(p_i), r_{\text{high}}(p_i)]$ $\forall 1 \leqslant i \leqslant m$.

We also construct a universal ball (of infinite radius) centered at the point that is the root of the tree formed by the λ-stretch hierarchical clustering of P, so that any point that is not in any of the balls constructed by the above rules lies in this ball.

3.3. Correctly answering $(1+\epsilon)$-approximate NN queries

In this subsection we prove that reporting the center of the smallest ball that contains a query point from the set of balls constructed by the algorithm ConstructBalls answers the approximate nearest neighbor query correctly.

Observation 3.2. $\|ab\| \geqslant 2r_{\text{loss}}(p)$ $\forall a \in$ Subtree(p) *and* $b \notin$ Subtree(p) *as* $2r_{\text{loss}}(p)$ *is the minimum separation between any point in the cluster from any point outside it by definition of* r_{loss}.

In order to limit the number of balls that we construct around a given point, we wish to determine the largest ball such that for any query point beyond that we can claim the existence of another point which is a near neighbor of the query point with a limited accumulated approximation error.

The following lemma establishes that $r_{\text{high}}(x)$ is a limit on the radius of the largest ball to be constructed around the point x, so that if the query point does not lie in this ball, then parent(x) is a near neighbor of the query point with some accumulated approximation error. Thus, we can ignore the point x at the expense of some accumulated approximation error.

By the definition of $r_{\text{high}}(x)$, the accumulated error is a factor of $1 + \gamma/(3\log n)$. This value is so chosen that repeated accumulation of approximation errors is bounded by a suitable value as we will see later.

Lemma 3.1. *For any query point q, if x is a $(1+\alpha)$-NN of q in P, and if $\|qx\| > r_{\text{high}}(x)$, then* parent$(x)$ *is a $((1+\gamma/(3\log n))(1+\alpha))$-NN of q in P.*

Proof. The proof of this lemma is similar to the proof of Lemma 2.13 in [11].

Let z be parent of x in the λ-stretch hierarchical clustering of the set of points P. Note that $\|qx\| > r_{\text{high}}(x) > r_{\text{death}}(x)$. By the same argument as in Lemma 2.10 of [11], $\|zx\| \leqslant 2n\lambda r_{\text{loss}}(x) = (\gamma/(3\log n))r_{\text{death}}(x) < (\gamma/(3\log n))\|qx\|$. Therefore, $\|zq\| \leqslant \|qx\| + \|zx\| \leqslant (1+\gamma/(3\log n))\|qx\| \leqslant (1+\gamma/(3\log n))(1+\alpha)d_P(q)$.

It follows that parent(x) is a $((1+\gamma/(3\log n))(1+\alpha))$-NN of q in P. □

In the following lemma, we show that it is possible to recursively consider the parent of the current candidate as the next nearest neighbor candidate if the query point does not lie in the largest ball around the current candidate. Of course, at every step that we shift the candidate to the parent, we accumulate an extra error in our approximation.

Lemma 3.2. *Let p be the NN of a query point q in P. Let r be the radius of the smallest ball containing q and let it be centered at x. If $r > r_{\text{high}}(\text{parent}^i(p))$ $\forall 0 \leqslant i \leqslant j-1$, then* parent$^j(p)$ *is a $((1+\gamma/(3\log n))^j$-approximate NN of q in P.*

Proof. The proof is by induction on j using Lemma 3.1. □

In the following lemma, we now show that since we are constructing balls in a manner that every ball is larger by a factor of $1+\gamma/3$ from the previously constructed ball, we may incur an additional approximation error factor of $1+\gamma/3$ in reporting the nearest neighbor. That is, that if a point p is an approximate nearest-neighbor of a query point q, such that q lies in one of the balls constructed in the ballset of p but we end up reporting another point x because the ball containing q centered at x is smaller than the ball containing q centered at p, even though p is closer (note that this is possible due to our construction of discrete set of balls), we only suffer an additional approximation error factor of $1+\gamma/3$, i.e., by reporting x instead of p.

Lemma 3.3. *For any query point q, if p is a $(1+\alpha)$-NN of q in P, and if there exists a ballset, centered at p, in $[r_{\text{low}}(t), r_{\text{high}}(t)]$ for some point t, and if the smallest ball containing q has radius r, such that $r_{\text{loss}}(t) \leqslant r \leqslant r_{\text{high}}(t)$ and is centered at x, then x is a $(1+\gamma/3)(1+\alpha)$-approximate NN of q in P.*

Proof. If $x = p$, then we are done.

Suppose $x \neq p$. Then two cases arise, either $q \notin B(p, r_{\text{high}}(t))$ or $q \in B(p, r_{\text{high}}(t))$. We will show in either case that $d_P(q) \leqslant \|xq\| \leqslant (1+\gamma/3)(1+\alpha)d_P(q)$.

Case I. $q \notin B(p, r_{\text{high}}(t))$.

In this case, $\|pq\| > r_{\text{high}}(t) \geqslant r > \|xq\|$. This implies that x is closer to q than p and therefore trivially $\|xq\| < (1+\gamma/3)(1+\alpha)d_P(q)$.

Case II. $q \in B(p, r_{\text{high}}(t))$.

Clearly, $q \notin B(p, r_{\text{low}}(t))$, because otherwise the smallest ball containing q would have radius $\leqslant r_{\text{low}}(t)$ and $r \geqslant r_{\text{loss}}(t) > r_{\text{low}}(t)$ leading to a contradiction that the smallest ball containing q has radius r. Hence, $\exists j$, such that $q \in B(p, r_j)$ and $q \notin B(p, r_{j+1})$, where $r_0 = r_{\text{low}}(t)$, implying that $r_j < \|pq\| \leqslant r_{j+1} = (1+\gamma/3)r_j$.

Therefore, $\|xq\| \leqslant r \leqslant r_{j+1} = (1+\gamma/3)r_j < (1+\gamma/3)\|pq\| \leqslant (1+\gamma/3)(1+\alpha)d_P(q)$.

Thus x is a $(1+\gamma/3)(1+\alpha)$-approximate NN of q in P. □

Suppose that the query point q is contained in the ball of radius $r_{\text{loss}}(p)$ for a point p. Then, clearly the nearest-neighbor of the query point q can only lie amongst the points that belong to the cluster formed by the subtree of the hierarchical clustering tree rooted at p. However, it still needs to be determined which of these points is closer, and p itself is also a candidate. This calls for the construction of more balls around p of radii corresponding to the ballsets

constructed around the child nodes of p. This way we determine which of p and its children is closer to the query point q. This is precisely the set of balls constructed by rule 2 in algorithm ConstructBalls. Note that we do not need to construct balls around p corresponding to the children further down the hierarchy, as the child nodes of p would then themselves be better candidates.

The following lemma shows that the construction of balls in algorithm ConstructBalls for the case described above leads to answering the queries with only a further compounded approximation error of a factor of $1+\gamma/3$.

Lemma 3.4. *For any query point q, if p is a $(1+\alpha)$-NN of q in P, and we have balls constructed around the points of P as defined by* ConstructBalls*, and if the smallest ball containing q has radius r, such that $r < r_{\text{loss}}(p)$ and is centered at x, then x is a $(1+\gamma/3)(1+\alpha)$-approximate NN of q in P.*

Proof. If $q \notin B(p, r_{\text{loss}}(p))$, then $\|pq\| > r_{\text{loss}}(p) > r \geqslant \|xq\|$ implying that x is closer to q than p and therefore trivially $\|xq\| \leqslant (1+\gamma/3)(1+\alpha)d_P(q)$.

Suppose $q \in B(p, r_{\text{loss}}(p))$. Since $q \in B(x, r)$ and $q \in B(p, r_{\text{loss}}(p))$, these balls intersect. Then $x \in \text{Subtree}(p)$ because all balls of *radius* $< r_{\text{loss}}(p)$ centered around points $\notin \text{Subtree}(p)$ cannot intersect since $\|ab\| \geqslant 2r_{\text{loss}}(p)$ $\forall a \in \text{Subtree}(p)$ and $b \notin \text{Subtree}(p)$. Let p_1 be the child of p, such that $x \in \text{Subtree}(p_1)$. We consider three cases based on the value of r.

Case I. $r < r_{\text{loss}}(p_1)$.

We know that $x \in \text{Subtree}(p_1)$ and $p \notin \text{Subtree}(p_1)$. Then $\|pq\| > r$, because otherwise $\|px\| \leqslant \|pq\| + \|qx\| < r + r = 2r \leqslant 2r_{\text{loss}}(p_1)$, and this would contradict Observation 3.2.

This implies that x is closer to q than p and therefore trivially $\|xq\| \leqslant (1+\gamma/3)(1+\alpha)d_P(q)$.

Case II. $r > r_{\text{high}}(p_1)$.

This case is not possible as we do not construct balls larger than $r_{\text{high}}(t)$ around any point t of P, and $r_{\text{high}}(t) \leqslant r_{\text{high}}(p_1)$ $\forall t \in \text{Subtree}(p_1)$, but we know that there is a ball centered at $x \in \text{Subtree}(p_1)$ that contains q.

Case III. $r_{\text{loss}}(p_1) \leqslant r \leqslant r_{\text{high}}(p_1)$.

Since there is a ballset around p in the interval $[r_{\text{low}}(p_1), r_{\text{high}}(p_1)]$, $\exists j$, such that $q \in B(p, r_j)$ and $q \notin B(p, r_{j+1})$, where $r_0 = r_{\text{low}}(p_1)$. This implies that $\|pq\| > r_j$. It must be that $r \leqslant r_{j+1}$, because otherwise $B(p, r_{j+1})$ would be a ball of radius smaller than r containing the query point q.

Therefore, $\|xq\| \leqslant r \leqslant r_{j+1} = (1+\gamma/3)r_j < (1+\gamma/3)\|pq\| \leqslant (1+\gamma/3)(1+\alpha)d_P(q)$.

Thus in all valid cases, x is a $(1+\gamma/3)(1+\alpha)$-approximate NN of q in P. □

In the following theorem, we combine the results of all the previous lemmas to show that the construction of balls in algorithm ConstructBalls does answer the approximate nearest-neighbor queries correctly.

Theorem 3.1. *For a point set P and an approximation factor ϵ, Let the smallest ball containing q, in the balls formed by algorithm* ConstructBalls(P, γ)*, where $\gamma = \epsilon/2$, be of radius r, centered at x, then x is a $(1+\epsilon)$-approximate NN of q in P.*

Proof. Let p be the NN of q in P.

Case I. $r < r_{\text{high}}(p)$.

- $r_{\text{loss}}(p) \leqslant r \leqslant r_{\text{high}}(p)$:
  If $r_{\text{loss}}(p) \leqslant r \leqslant r_{\text{high}}(p)$, then by the construction in algorithm ConstructBalls, since there exists a ballset in $[r_{\text{low}}(p), r_{\text{high}}(p)]$, using Lemma 3.3, x is a $(1+\gamma/3)$-NN of q in P.

– $r < r_{\text{loss}}(p)$:
If $r < r_{\text{loss}}(p)$, then using Lemma 3.4, x is a $(1+\gamma/3)$-NN of q in P.

Case II. *$r > r_{\text{high}}(\text{parent}^i(p))$ and $r < r_{\text{high}}(\text{parent}^{(i+1)}(p))$ for some i.*

By Lemma 3.2, $\text{parent}^{i+1}(p)$ is a $((1+\gamma/(3\log n))^{i+1}$-approximate NN of q in P.

– If $r_{\text{loss}}(\text{parent}^{i+1}(p)) \leqslant r \leqslant r_{\text{high}}(\text{parent}^{i+1}(p))$, then by the construction in algorithm ConstructBalls, since there exists a ballset in $[r_{\text{low}}(\text{parent}^{i+1}(p)), r_{\text{high}}(\text{parent}^{i+1}(p))]$, centered at $\text{parent}^{i+1}(p)$. Using Lemma 3.3, x is a $((1+\gamma/(3\log n))^{i+1}(1+\gamma/3)$-approximate NN of q in P.
– If $r < r_{\text{loss}}(\text{parent}^{i+1}(p))$, then using Lemma 3.4, x is a $((1+\gamma/(3\log n))^{i+1}(1+\gamma/3)$-NN of q in P.
Also $i+1 \leqslant \log n$, since height of the tree formed by the λ-stretch hierarchical clustering of P is bound by $\log n$.
Therefore x is a $((1+\gamma/(3\log n))^{\log n}(1+\gamma/3)$-approximate NN of q in P.

Case III. *$r > r_{\text{high}}(t)$, where t is the root of the tree formed by the λ-stretch hierarchical clustering of P.*

There is no ball in P of radius $> r_{\text{high}}(p)$. Thus t is reported as the approximate-NN of q in P as q lies in the universal ball centered at t.

By Lemma 3.2, t is $((1+\gamma/(3\log n))^i$-approximate NN of q in P, where $i \leqslant \log n$, since height of the tree formed by the λ-stretch hierarchical clustering of P is bound by $\log n$.

Therefore x is a $((1+\gamma/(3\log n))^{\log n}(1+\gamma/3)$-approximate NN of q in P.

And as $(1+\gamma/3)(1+\gamma/(3\log n))^{\log n} \leqslant 1+2\gamma$, x is a $(1+2\gamma)$-approximate NN of q in P.

Hence x is a $(1+\epsilon)$-approximate NN of q in P. □

3.4. A bound on the total number of balls required

In the following theorem we analyze the number of balls that we construct in the algorithm ConstructBalls.

Theorem 3.2. *The total number of balls constructed by the algorithm* ConstructBalls$(P, \epsilon/2)$ *is* $O((1/\epsilon^2)n\log(n/\epsilon))$.

Proof. The number of balls in a ball set is $O((1/\epsilon)\log_{1+O(\epsilon)}(n/\epsilon)) = O((1/\epsilon^2)\log(n/\epsilon))$.
For each point, one ballset is constructed for the point by rule 1 of algorithm ConstructBalls. Additionally, one ballset is constructed for each child of the point by rule 2 of algorithm ConstructBalls. By charging the balls constructed around the child nodes (using rule 2) by algorithm ConstructBalls to the respective child nodes, the charge incurred at each point is at most that of 2 ball sets, i.e., $O((1/\epsilon^2)\log(n/\epsilon))$.

Therefore, the total charge incurred over the n points is $O((1/\epsilon^2)n\log(n/\epsilon))$. □

This improves the bound on the number of balls constructed in [11] by a factor of $\log n$. These balls can be used to construct cells, i.e., quadtree boxes, and then store them in a BBD-tree as described in [11] resulting in an improvement of a factor of $\log n$ in space complexity and preprocessing time while keeping the query time logarithmic.

4. A linear space solution based on ε-PLSB and a new hierarchical clustering

In the previous section, we saw that the recursive structure of the Har-Peled's [11] data structure can be removed to yield an $O(n\log(n))$ space. The extra $\log(n)$ factor is appearing because, around each point of the database, we construct balls of radius between $r_{\text{loss}}(p), r_{\text{loss}}(p)\cdot(1+\epsilon), r_{\text{loss}}(p)\cdot(1+\epsilon)^2, \ldots, r_{\text{death}}(p)$. Since, we have defined $r_{\text{death}}(p)$ to be $\Omega(n/\varepsilon)$ times $r_{\text{loss}}(p)$, this contributes $\Omega(\log(n))$ for each point in the database. But, actually we do not need $r_{\text{death}}(p)$ to be as high as that. We observe that r_{death} is required to be the distance beyond which any point of the subtree(p) is ε nearest neighbor of the query point q. An appropriate distance for that could be $O(\text{diam}(\text{subtree}(\text{parent}(p)))/\varepsilon)$, where diam denotes the diameter of a point set.

Unfortunately, even with this definition of r_{death}, the number of balls generated could be $O(n\log(n))$. To see that, consider the example of n points (say $p_1, p_2, \ldots, p_n$) all lying a line such that $\text{dist}(p_i, p_{i+1}) = 1 + ih$, where h is

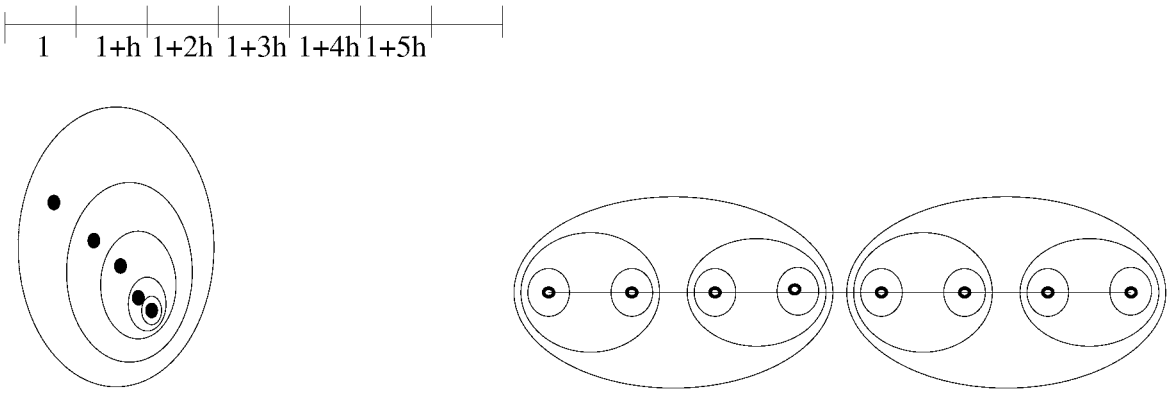

Fig. 2. Two different clustering for almost equidistant points in a line.

an infinitesimally small number. Clearly, the MST is a single chain. In this case, the number of balls generated by algorithm in [11] (and its modification here) is $\Omega(n \log(n))$, even if we redefine r_{death} as suggested. The problem is that the algorithm in [11] disregards the edge-lengths of the approximate MST, except for their *relative ordering* when constructing the hierarchical clustering. To solve that problem, we construct the hierarchical clustering in a different manner. We also give importance to the *magnitudes* of the edge lengths of the MST in addition to the relative ordering, namely, we do not distinguish between *nearly equal* edge lengths. Consequently, in the previous example the hierarchical clustering may take different forms as in Fig. 2. The second one is clearly preferred (we shall return to this example later).

4.1. ε-NNS and ε-PLSB

The problem that we address here is a related problem called *Approximate Point Location in Smallest Ball* (ε-PLSB) which is an approximate version of *Point Location in Smallest Ball* (PLSB) problem. In defining ε-PLSB, we take care of the following.

(1) The approximate version of PLSB can be solved by searching for cubes in a hierarchical grid as [11] (see Appendix C).
(2) The approximate nearest neighbor searching problem is reducible to the ε-PLSB problem.

For this, we make use of another point q' which will ensure that the approximation factor $1+\varepsilon$ is achieved from the nearest neighbor of p. A point p is an ε-PLSB solution for a query point q, if there exists q' such that (i) p is a *PLSB* solution for q' *and* (ii) q' is *near* q. The formal definitions follow.

Definition 4.1.

(1) **PLSB.** Given a set of points P, and a finite set $Q \subset P \times \mathbb{R}$, we want to build a data structure D, such that given any query point q, we are able to return a pair $(p, r) \in Q$, such that $b(p, r)$ is the smallest ball containing q. If there is no such ball, then it should return NIL.
(2) **ε-PLSB.** Given a set of points P, and a finite set $Q \subset P \times \mathbb{R}$, we want to build a data structure D, such that given any query point q:
   (i) If $q \in b(p, r)$ for some $(p, r) \in Q$, then we must return a pair $(p', r') \in Q$, such that $r' \leqslant r$ and there exists a point $q' \in b(p', r')$ such that $\text{dist}(q, q') \leqslant r'\varepsilon$ and $b(p', r')$ is the smallest ball containing q' among balls from $\{b(p, r) \mid (p, r) \in Q\}$ (p' is solution to PLSB query for q').
   (ii) If q is not contained in any ball from the set $\{b(p, r(1+\varepsilon)) \mid (p, r) \in Q\}$, then we should return NIL.
   (iii) Otherwise, we can return anything (even NIL).

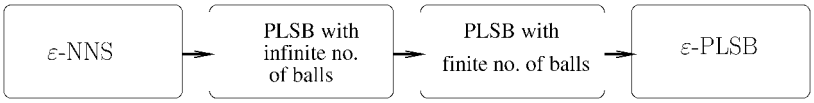

Fig. 3. The reduction of ε-NNS to ε-PLSB—the intermediate steps illustrate the proof strategy.

Given a set of points P, we will construct a *special* set of balls B around points in P (cf. Definition 4.5), so that the answer to ε-PLSB in B yields the approximate nearest neighbor in P. For that, we follow strategy described by Fig. 3.

The first step in the reduction is accomplished by using a carefully constructed hierarchical clustering of the point set P. The definition of our hierarchical clustering is actually quite generic. It is a binary tree with singleton subsets of P (the input point set) as leaves. The internal nodes are denoted by the union of leaves of the sub-tree rooted at that node. Given *any* such tree, we construct an instance I of ε-PLSB, such that any nearest neighbor query for the point set P can be solved by solving the query q for I. For two sets X and Y, we denote $\min_{x\in X, y\in Y} \text{dist}(x, y)$ by $\text{dist}(X, Y)$. Using this notation, we define the hierarchical clustering as follows.

Definition 4.2 *(Hierarchical clustering).* Given a point set P, its hierarchical clustering is a *binary* tree with singleton subsets of P as leaves (each singleton subset of P appearing exactly once). The internal nodes are denoted by the union of the leaves of the sub-tree rooted at them. Each node in the tree is thus a subset of P and is called a cluster. Note that the root is the cluster P. Each internal node v of the tree is tagged with two values $r_{\min}(v)$ and $r_{\max}(v)$ with the following properties.

(1) $r_{\min}$ values increase from leaves to root.
(2) $r_{\min}(v) \leqslant \frac{1}{2}\,\text{dist}(\text{left}(v), \text{right}(v))$.
(3) $r_{\max}(v) \geqslant 8\frac{\text{diam}(v)}{\varepsilon}$.

Definition 4.3. For every cluster v of T, we pick an arbitrary point $p \in v$ and call it leader(v).

Definition 4.4 *(*BallRange(p, r_1, r_2)*).* It is defined to be the set of all balls centered at p of radius between r_1 and r_2. $\text{BallRange}(p, r_1, r_2) = \{b(p, r) | r_1 \leqslant r \leqslant r_2\}$. Note that there are infinite number of balls in $\text{BallRange}(p, r_1, r_2)$.

Definition 4.5 *(*BallSet(v)*).* For a cluster v of a hierarchical clustering T, we define BallSet(v) to be

- If v is leaf, then $\text{BallSet}(v) = \phi$.
- Otherwise, $\text{BallSet}(v) = \text{BallSet}(\text{left}(v)) \cup \text{BallSet}(\text{right}(v)) \cup B_l \cup B_r$, where $B_l = \text{BallRange}(\text{leader}(\text{left}(v)), r_{\min}(v), r_{\max}(v))$, $B_r = \text{BallRange}(\text{leader}(\text{right}(v)), r_{\min}(v), r_{\max}(v))$.

Note that BallSet(v) is actually a union of $2|v| - 2$ BallRanges.

Definition 4.6 *(Inner ball versus outer ball).* A ball is called an inner ball, if it is the smallest ball for one of the BallRanges of BallSet(v). Otherwise, it is called an outer ball.

Observation 4.1. *If* $b(p, r) \in \text{BallSet}(v)$ *is an inner ball of* BallSet(v)*, then* $r = r_{\min}(u)$*, for* $u \in \text{Subtree}(v)$ *in the hierarchical clustering of* v.

Observation 4.2. *Suppose, the ball* $b(p, r)$ *is the smallest ball in* BallSet(v) *containing* q*. If* $b(p, r)$ *is an outer ball, then* $\text{dist}(q, p) = r$.

4.2. Details of the reduction

Given a hierarchical clustering, we give the strategy of reduction from ε-NNS to ε-PLSB for an l_p metric on R^d. This process is hypothetically divided in three steps. In the first step, we generate an infinite set of balls P_1, such that

result of PLSB query q to P_1 is a ball, centered around an $\varepsilon/7$-nearest neighbor of q. In the next step, we generate a finite set of balls P_2 from P_1, such that result of PLSB query q to P_2 is a ball centered around $2\varepsilon/3$-nearest neighbor of q. Finally, we prove that the result of $\varepsilon/7$-PLSB query q to P_2 is a ball centered around ε-nearest neighbor of q in P.

4.2.1. Generating P_1 (reduction to PLSB with infinite No. of balls)

Here we have to reduce $\varepsilon/7$-NNS to PLSB with infinite number of balls. For that, we prove that for any cluster v of T, BallSet(v) is such a point set. The exact statement is the Theorem 4.1.

Theorem 4.1. *For a cluster v of T, given a query point q, a point $p \in v$ is $\varepsilon/7$ nearest neighbor of q among points of v, if one of the following is true.*

- *If p is the center of the smallest ball in* BallSet(v) *containing q.*
- *There is no ball in* BallSet(v) *containing q.*

Note that the second condition depends only on q, and so any arbitrary database point p is $\varepsilon/7$-nearest in that case.

Proof. We prove it by induction on the height of v in T. In the base case, v is a leaf, and BallSet(v) is empty. So, it is trivially proved.

Now, in the induction step, we assume the result for left(v) and right(v), and prove it for v. Let $p_1 = \text{leader}(\text{left}(v))$ and $p_2 = \text{leader}(\text{right}(v))$. By definition, $\text{BallSet}(v) = (\text{BallSet}(\text{left}(v)) \cup B_l) \cup (\text{BallSet}(\text{right}(v)) \cup B_r)$, where B_l and B_r are a set of balls around p_1 and p_2, respectively. Note that $\text{BallSet}(\text{left}(v)) \cup B_l = B_1$ (say) is a set of balls with centers among points of left(v) and $\text{BallSet}(\text{right}(v)) \cup B_r = B_2$ (say) is a set of balls with centers among points of right(v). We consider three cases (exhaustive but not necessarily disjoint).

Case 1. Since, $b(p_1, r_{\max}(v)) \in B_1$, so $\text{dist}(q, p_1) \geqslant r_{\max}(v) \geqslant 8\,\text{diam}(v)/\varepsilon$. So, distance between q and p_1 is large compared to the diameter of v. It is easy to see that for any point $p \in v$, $\frac{\text{dist}(q,p)}{\text{dist}(q,v)} \leqslant \frac{r_{\max}(v)}{r_{\max}(v)-\text{diam}(v)} \leqslant \frac{8}{8-\varepsilon} \leqslant 1 + \varepsilon/7$.

Case 2. There exists no ball in B_2 containing q: Similar to Case 1.

Case 3. There exists a ball each in B_1 and B_2 containing q: Let, the smallest ball in B_1 and B_2 containing q be $b(p_1^*, r_1)$ and $b(p_2^*, r_2)$, respectively. To handle this case, we use the Lemma 4.1 proved later.

Lemma 4.1. *p_1^* is an $\varepsilon/7$-nearest neighbor of q, among points of* left(v)*, and p_2^* is an $\varepsilon/7$-nearest neighbor of q, among points of* right(v).

Now we consider the three sub-cases arising in Case 3. We have $\text{dist}(q, p_1^*) \leqslant r_1$ and $\text{dist}(q, p_2^*) \leqslant r_2$.

Case 3a. The ball $b(p_1^*, r_1)$ is an inner ball: In this case, from Observation 4.1, we have $r_1 = r_{\min}(v')$ for some $v' \subseteq v$. Using this with the definition of $r_{\min}$ we get: $\text{dist}(q, p_1^*) \leqslant r_1 = r_{\min}(v') \leqslant r_{\min}(v) \leqslant 0.5\,\text{dist}(\text{left}(v), \text{right}(v))$. This implies that $\text{dist}(q, \text{left}(v)) \leqslant 0.5\,\text{dist}(\text{left}(v), \text{right}(v))$. But, $\text{dist}(q, \text{right}(v)) \geqslant \text{dist}(\text{left}(v), \text{right}(v)) - \text{dist}(q, \text{left}(v))$. Therefore, $\text{dist}(q, \text{right}(v)) \geqslant 0.5\,\text{dist}(\text{left}(v), \text{right}(v)) \geqslant \text{dist}(q, \text{left}(v))$. So, $\text{dist}(q, v) = \text{dist}(q, \text{left}(v))$. Now, $\frac{\text{dist}(q,p_1^*)}{\text{dist}(q,v)} = \frac{\text{dist}(q,p_1^*)}{\text{dist}(q,\text{left}(v))} \leqslant 1 + \varepsilon/7$. So, p_1^* is an $\varepsilon/7$ nearest neighbor of q in v.

Also, any ball in B_2 containing q must have a radius larger than $\text{dist}(q, \text{right}(v))$. But, $\text{dist}(q, \text{right}(v)) \geqslant 0.5\,\text{dist}(\text{left}(v), \text{right}(v)) \geqslant r_{\min}(v') = r_1$. So, any ball in B_2 containing q is larger than $b(p_1^*, r_1)$. So, we see that the smallest ball containing q in BallSet(v) is centered around $\varepsilon/7$ nearest neighbor of q in v.

Case 3b. The ball $b(p_2^*, r_2)$ is an inner ball: Similar to Case 3a.

Case 3c. Both the balls $b(p_2^*, r_2)$ and $b(p_2^*, r_2)$ are outer balls: From Observation 4.2, we have $r_1 = \text{dist}(q, p_1^*)$ and $r_2 = \text{dist}(q, p_2^*)$. Without loss of generality, $r_1 \leqslant r_2$. So, $b(p_1^*, r_1)$ is the smallest ball containing q, in $B_1 \cup B_2 = \text{BallSet}(v)$.

Also, $\frac{\text{dist}(q,p_1^*)}{\text{dist}(q,\text{left}(v))} \leqslant 1 + \varepsilon/7$ and $\frac{\text{dist}(q,p_2^*)}{\text{dist}(q,\text{right}(v))} \leqslant 1 + \varepsilon/7$ from the Lemma 4.1. Since, p_1^* is nearer to q than p_2^*, therefore $\frac{\text{dist}(q,p_1^*)}{\text{dist}(q,\text{right}(v))} \leqslant 1 + \varepsilon/7$. Therefore $\frac{\text{dist}(q,p_1^*)}{\min(\text{dist}(q,\text{left}(v)),\text{dist}(q,\text{right}(v)))} \leqslant 1 + \varepsilon/7$. This implies that p_1^* is the $\varepsilon/7$ nearest neighbor of q in v. □

Now we give the proof of Lemma 4.1.

Proof. We shall prove the Lemma 4.1 only for B_1. The proof for B_2 is similar. We have $B_1 = \text{BallSet}(\text{left}(v)) \cup B_l$. From induction hypothesis, if no ball in BallSet(left(v)) contains q, then all the points are $\varepsilon/7$-nearest neighbor of q in left(v). If q is contained in a ball of BallSet(left(v)), then the smallest ball in BallSet(left(v)) containing q, is centered around an $\varepsilon/7$-nearest neighbor of q in left(v). Suppose $b(p', r')$ is the smallest ball containing q in BallSet(left(v)). If this is also the smallest ball in B_1, then we are done. Suppose it is otherwise. Then the smallest ball in B_1 is from B_l. Suppose, $b(p_1, r'')$ is the smallest ball in B_l (and hence in B_1).

Case (i). $b(p', r')$ is an outer ball: In this case, $\text{dist}(q, p') = r'$. So, if the smaller ball is $b(p_1, r'')$, then $\text{dist}(q, p_1) \leqslant r'' \leqslant r' = \text{dist}(q, p')$. So, we choose a ball nearer than the $\varepsilon/7$-nearest neighbor. Clearly, that is also $\varepsilon/7$-nearest neighbor.

Case (ii). $b(p', r')$ is an inner ball: In this case, r' is $r_{\min}(v')$ for some $v' \subset v$. The smallest ball in B_l is of radius $r_{\min}(v)$. So $r'' \geqslant r_{\min}(v) \geqslant r_{\min}(v') = r'$. So, the $b(p', r')$ is a smaller ball, which is a contradiction. So, this proves Lemma 4.1. □

4.2.2. Generating P_2 from P_1 (reduction to PLSB with finite No. of balls)

In this section, we show that we need to retain only a finite number of balls of P_1. For that we consider each ball range separately.

Theorem 4.2. *Given a hierarchical clustering $T = (V, E)$ for a point set P, we can construct a set of*

$$2\sum_{v\in V} \max\left(1, \log_{1+\varepsilon} \frac{r_{\max}(v)}{r_{\min}(v)}\right)$$

balls such that following is true for a PLSB constructed for them. Given a query point q, the point returned by PLSB data structure is $1 + 2\varepsilon/5$ nearest neighbor of q among points of P.

Proof. We start with BallSet(P). Recall that there are infinite number of balls. We shall retain a carefully selected finite set from these. Recall that balls in the BallSet(P) can be expressed as union of $2|P| - 2$ BallRanges. We replace each maximal $\text{BallRange}(p, r_1, r_2) \subset \text{BallSet}(P)$ with

$$\left\{ b(p, r) \;\middle|\; r = r_1\left(1 + \frac{\varepsilon}{7}\right)^k \wedge r \leqslant r_2(1 + \varepsilon/7) \wedge k \in \{0, 1, 2, \ldots\} \right\}.$$

Let B be this finite set of balls. We will show that, a PLSB constructed for B returns a ball which is centered around a $(1 + \varepsilon/7)(1 + \varepsilon/7)$ ($\leqslant (1 + 2\varepsilon/5)$) nearest neighbor. This completes the proof of the theorem.

Suppose, q is a query point, for $(\varepsilon/7)$-PLSB of B. Suppose, $b(p, r)$ is the smallest ball containing q among balls of BallSet(P).

Case 1 *($b(p, r)$ is an inner ball of BallSet(P)).* From Theorem 4.1, we know, that p is an $\varepsilon/7$-nearest neighbor of q. Also, we know that $b(p, r)$ is contained in B(B contains all the inner balls of BallSet(P)). So, $b(p, r)$ is the smallest ball in B containing q, and p is an $\varepsilon/7$-nearest neighbor of q. Hence proved.

Case 2 *($b(p,r)$ is an outer ball of* BallSet(P)*).* In this case, we observe that there is a ball $b(p,r') \in B$, with r' at most $r(1+\varepsilon/7)$ which contains q. So, the smallest ball in B containing q is of radius utmost $r(1+\varepsilon/7)$. So, the center of the smallest ball in B containing q is at a distance of at most $r(1+\varepsilon/7)$ from q, which is $(1+\varepsilon/7)^2\,\mathrm{dist}(q,P)$. □

4.2.3. Reduction to $\varepsilon/7$-PLSB with finite No. of balls

Theorem 4.3. *Given a hierarchical clustering $T=(V,E)$ for a point set P, we can construct a set of*

$$2\sum_{v\in V}\max\left(1,\log_{1+\varepsilon}\frac{r_{\max}(v)}{r_{\min}(v)}\right)$$

balls such that following is true for an $\varepsilon/7$-PLSB constructed for it. Given a query point q, the point returned by $\varepsilon/7$-PLSB data structure is $1+\varepsilon$ nearest neighbor of q among points of P.

Proof. We use the same finite set of balls B as used in Theorem 4.2, and show that an $\varepsilon/7$-PLSB made for it satisfies the above condition. Suppose, $b(p,r)$ is the ball returned by the $\varepsilon/7$-PLSB data structure for B.

Case 1 *($q \in b(p,r)$).* From definition of $\varepsilon/7$-PLSB, we get that $b(p,r)$ is the smallest ball in B containing q. Using Theorem 4.2, we get that p is $(1+\varepsilon/7)^2$ nearest neighbor of q.

Case 2 *($q \notin b(p,r)$).* In this case, we observe from the definition of $\varepsilon/7$-PLSB, that there exists a query point q', such that $\mathrm{dist}(q,q') \leqslant \varepsilon r/7$ and $b(p,r)$ is the smallest ball in B containing q'. Using this with Theorem 4.2, $\mathrm{dist}(q,P) + \mathrm{dist}(q,q') \geqslant \mathrm{dist}(q',p)/(1+\varepsilon/7)^2$. This implies $\mathrm{dist}(q,P) \geqslant \mathrm{dist}(q',p)/(1+\varepsilon/7)^2 - \varepsilon r/7$. Simplifying, we get $\mathrm{dist}(q,P) \geqslant \mathrm{dist}(q',p)(1+\varepsilon)$. So, p is a ε nearest neighbor of q. □

B is composed of ball ranges, two for each cluster. Number of balls in ballranges corresponding to cluster v is $2\max(1,\log_{1+\varepsilon}\frac{r_{\max}(v)}{r_{\min}(v)})$. So, we need at most $2\sum_{v\in V}\max(1,\log_{1+\varepsilon}\frac{r_{\max}(v)}{r_{\min}(v)})$ balls.

4.3. Some observations on MST based hierarchical clustering

Given a point set P, we want to construct a hierarchical clustering such that

$$2\sum_{v\in V}\max\left(1,\log_{1+\varepsilon}\frac{r_{\max}(v)}{r_{\min}(v)}\right) = O\big(|P|\log_{1+\varepsilon}(1/\varepsilon)\big).$$

The procedure to construct such a hierarchical clustering is quite involved. We shall first describe a simpler hierarchical clustering which achieves $O(|P|\log_{1+\varepsilon}(|P|/\varepsilon))$ bound. Then we describe methods for improving it to obtain $O(|P|\log_{1+\varepsilon}(1/\varepsilon))$ bound.

The first hierarchical clustering that we describe is directly based on the MST of the point set. Suppose, the MST of the point set P is $M=(P,E'_M)$. Then we construct the hierarchical clustering as follows: The longest edge in E'_M divides P into two parts, say P_1 and P_2. We find the hierarchical clusterings of P_1 and P_2 recursively, say C_1 and C_2. The MST hierarchical clustering of P, then, has P as root and C_1 and C_2 as two children of the root.

Suppose, v is a cluster in the MST hierarchical clustering. We use

$$r_{\min}(v) = \frac{1}{2}\,\mathrm{dist}\big(\mathrm{left}(v),\mathrm{right}(v)\big) \quad \text{and}$$
$$r_{\max}(v) = 8\,\mathrm{diam}(v)/\varepsilon.$$

Consider any two points $p_1, p_2 \in v$. Consider the path connecting p_1 and p_2 in the MST. Suppose, $e_1, e_2, \ldots, e_l$ are the edges in the path. Then by repeated application of triangle inequality, we get $\mathrm{dist}(p_1,p_2) \leqslant \mathrm{len}(e_1)+\mathrm{len}(e_2)+\cdots+\mathrm{len}(e_l)$. Since, l is at most n and $\mathrm{len}(e_i) \leqslant \mathrm{dist}(\mathrm{left}(v),\mathrm{right}(v))$ (because $\mathrm{dist}(\mathrm{left}(v),\mathrm{right}(v))$ is the length of the largest edge in MST over v). Therefore, for any $p_1, p_2 \in v$ we have $\mathrm{dist}(p_1,p_2) \leqslant n\,\mathrm{dist}(\mathrm{left}(v),\mathrm{right}(v))$. So, diameter of v is at most $n\,\mathrm{dist}(\mathrm{left}(v),\mathrm{right}(v))$. Thus,

$$r_{\max}(v) \leqslant 8n\,\mathrm{dist}\big(\mathrm{left}(v),\mathrm{right}(v)\big)/\varepsilon.$$

So,

$$\frac{r_{\max}(v)}{r_{\min}(v)} = O(n/\varepsilon).$$

Thus

$$2\sum_{v\in V} \max\left(1, \log \frac{r_{\max}(v)}{r_{\min}(v)}\right) = O\big(|P| \log_{1+\varepsilon}(|P|/\varepsilon)\big).$$

The problems with MST hierarchical clustering are the following:

(1) The preprocessing involves construction of the MST, which requires nearly quadratic time.
(2) The preprocessing involves finding diameters of point sets which is computationally expensive.
(3) The space requirement is not linear but $n \log n$.

To handle the first problem, we use approximate MSTs. To handle the second problem, we use alternative definitions of $r_{\max}(v)$. With regards to the third problem, let us look at some examples before we propose our solution in the next subsection.

For the existing MST hierarchical clustering, the space requirement is $\Omega(|P| \log(|P|))$ in the worst case. For example, consider a set of n points in a straight line, $p_1, p_2, \ldots, p_n$, with $\text{dist}(p_i, p_{i+1}) = 1 + ih$. Here h is an infinitesimally small quantity. In this case, the MST is a line graph (Fig. 2) and in this case

$$\sum_p \log \frac{r_{\max}(p)}{r_{\min}(p)} = O(\log 1 + \log 2 + \cdots + \log n) = \Omega(n \log n).$$

For the case where the edge lengths are growing geometrically, say, 2^i, for $i = 1, 2, \ldots$, the MST hierarchical clustering is actually linear.

4.4. An alternate hierarchical clustering

For exposition, we describe a linear space solution for points in a line case (i.e. one-dimensional case).

In the previous example, let us use the observation that all the edges in the MST are *almost* equal and ignore the strict ordering. Using this idea, we construct the hierarchical clustering as in Fig. 2 (the second clustering), in form of a fully balanced tree. In this case,

$$\sum_v \log \frac{r_{\max}(v)}{r_{\min}(v)} = O\left(\frac{n}{2}\log 2 + \frac{n}{4}\log 4 + \cdots + \log n\right) = O(n).$$

The main idea in our solution is to partition the edges of this MST into sets, such that all the edges within a set have lengths within factor of two of each other. While constructing the hierarchical clustering from the approx MST, all the edges within a single partition are considered as if of same length. In the usual MST hierarchical clustering, we sort the edges according to their lengths, and keep on merging clusters connected by the edges, considering the edges in increasing order of their length. Here, we will reorder the merging sequence. For the general d-dimensional clustering, this reordering is much more involved. However, for one dimension, we present a simpler strategy.

Let $T = (P, E_M)$ be the MST for P. We partition E_M, into $\{E_k \mid k \in \mathbb{Z}\}$, where $E_k = \{e \in E_M \mid 2^k \leqslant \text{len}(e) < 2^{k+1}\}$. Suppose $E_{k_1}, E_{k_2}, \ldots, E_{k_l}$, are the set of all *non-empty* E_{k_i}'s with $k_i < k_{i+1}$ (Fig. 4). We consider all edges in each E_{k_i} as being of effectively the same length and try to keep the clustering more balanced. Suppose the clusters $c_0, c_1, c_2, \ldots, c_{k-1}$ are to be merged into one due to edges in E_{k_i}. Then, these clusters would form a contiguous interval on the real line (follows from the fact that the MST is a line graph). Without loss of generality, we assume that the order $c_0, c_1, c_2, \ldots, c_{k-1}$ is same as that appears on the real line. Then we form any *height balanced binary* tree with $c_0, c_1, \ldots, c_{k-1}$ as leaves in order. The merge is done according to the natural order suggested by this tree. For this clustering, the $r_{\min}(v)$ for any cluster v created in phase i is equal to $2^{k_i - 1}$.

4.4.1. Proof for points in a line case

The result that the generated clustering is linear follows from following two observations.

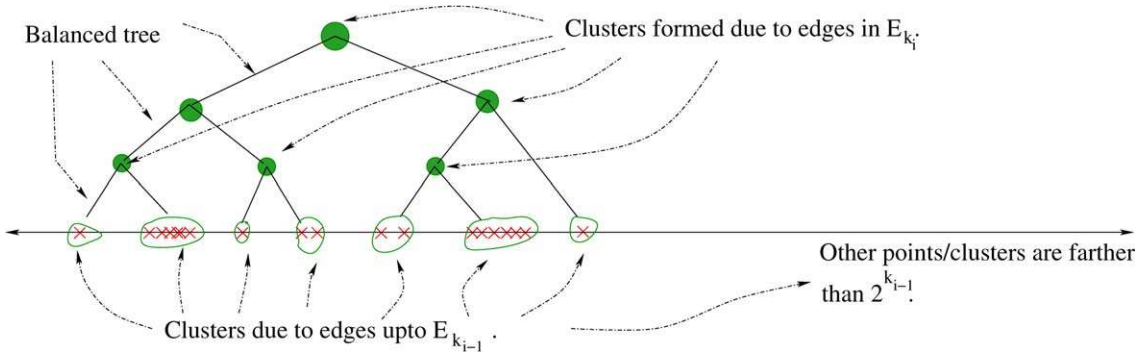

Fig. 4. Linear Hierarchical clustering for points in a line.

(1) If W_i is the set of clusters that we add when we consider edges in E_{k_i} (so-called phase i), then

$$\sum_{v \in W_i} \log \frac{r_{\max}(v)}{r_{\min}(v)} = O\left(\sum_{e \in E_{k_1} \cup E_{k_2} \cup \cdots \cup E_{k_i}} \frac{\text{len}(e)}{2^{k_i}} \right).$$

This result requires some technical details that we provide for the general d-dimensional case.

(2) The size of hierarchical clustering is linear. This follows by observing that in the previous result, if we add the contribution of each edge $e \in E_{k_i}$ to the total sum across all phases, we obtain $O(\text{len}(e) \sum_{j=i}^{\infty} (\frac{1}{2^{k_i}})) = O(1)$ as $\text{len}(e) = O(2^{k_i})$. Since there are $O(|P|)$ edges in the MST, the total sum is $(O(|P|))$.

4.5. Constructing the general hierarchical clustering

As in the [11], we use a λ-MST instead of an exact MST.

The central idea in our solution is to partition the edges of a λ-MST into sets, such that all the edges within a set have lengths within a factor of two of each other (actually any constant factor would work). While constructing the hierarchical clustering from the approximate MST, all the edges within a single set are considered as if of same length.

Let $T = (P, E_M)$ be a λ-MST for P. We partition E_M, into $\{E_k \mid k \in \mathbb{Z}\}$, where $E_k = \{e \in E_M \mid 2^k \leqslant \text{len}(e) < 2^{k+1}\}$. Suppose $E_{k_1}, E_{k_2}, \ldots, E_{k_l}$, be the set of all *non-empty* E_k's with $k_i < k_{i+1}$.

Recall that a hierarchical clustering is a labeled tree with the vertex set as a subset of power set of P. We construct the hierarchical clustering bottom up in l phases (l is the number of non-empty E_i's). In any phase i, we start with a forest[1] F_i. We initialize with the singletons i.e. $F_1 = (\{\{p\} \mid p \in P\}, \phi)$. Then, we iteratively *merge* the trees of F by creating new clusters in the forest, until we finally get a hierarchical clustering for P. By merging of two trees with roots v_1 and v_2, we mean adding a new cluster $v_1 \cup v_2$, with v_1 and v_2 as its two children. For merging more than two trees, we mean to keep on doing the binary merge until they become part of a single tree. F_i is the forest at the start of phase i. So, $F_1 = (\{\{p\} \mid p \in P\}, \phi)$. We maintain the invariant that, C is a root of a tree in F_i iff C is a maximal connected component of $(P, E_{k_1} \cup E_{k_2} \cup \cdots \cup E_{k_{i-1}})$.

For any cluster v added in phase i, we assign

$$r_{\min}(v) = \frac{2^{k_i}}{2\lambda}.$$

Observation 4.3.

(1) $r_{\min}$ *values increase from leaves to root. This is because, vertices added in later phases* (*which have larger* $r_{\min}$ *values*) *are always "above" the vertices in earlier phases, in the hierarchical clustering.*

(2) $r_{\min}(v) \leqslant \frac{1}{2} \text{dist}(\text{left}(v), \text{right}(v))$. *The proof is as follows. First observe that, the* λ*-MST edge going across any two maximal connected components of* $(P, E_{k_1} \cup E_{k_2} \cup \cdots \cup E_{k_{i-1}})$ *is of length at least* 2^{k_i}. *So, the distance*

[1] A forest is a graph, which can be expressed as a union of vertex-disjoint trees.

between those components is at least $\frac{2^{k_i}}{\lambda}$. *Since, children of any cluster added in phase* i *is also a union of maximal connected components of* $(P, E_{k_1} \cup E_{k_2} \cup \cdots \cup E_{k_{i-1}})$, *the distance between them is also lower bounded by* $\frac{2^{k_i}}{\lambda}$.

Definition 4.7 *(*$\mathrm{Path}_T(p_1, p_2)$*).* Given a λ-MST T, we define $\mathrm{Path}_T(p_1, p_2)$ for two points p_1 and p_2 as the set of edges on the unique path on T connecting p_1 and p_2.

Definition 4.8 *(*$\mathbb{S}(v)$*).* Given a λ-MST T, for a $v \subseteq P$, we define

$$\mathbb{S}(v) = \bigcup_{p_1, p_2 \in v} \mathrm{Path}_T(p_1, p_2).$$

Note that if v spans[2] a connected component in T, then $\mathbb{S}(v)$ is simply the set of edges in T, which go across vertices of v.

Definition 4.9 *(*$S(v)$*).* We define $S(v)$ to be sum of lengths of all the edges in $\mathbb{S}(v)$.

In our hierarchical clustering, we shall use

$$r_{\max}(v) = 8\frac{S(v)}{\varepsilon}.$$

Observation 4.4.

$$\mathrm{diam}(v) \leqslant S(v).$$

The proof is a simple application of triangle inequality along the path joining the two most distant points in v.

From Observations 4.4 and 4.3, we conclude that the $r_{\min}$ and $r_{\max}$ values are well defined (consistent with the definition of the hierarchical clustering).

In what follows, we let $\alpha = \log_{1+\varepsilon}(\frac{\lambda}{\varepsilon})$.

Theorem 4.4. *Given a point set* P *and a* λ*-MST (say* T*) of* P*, there exists a hierarchical clustering of* P*, with set of clusters* V *such that* $2\sum_{v \in V} \max(1, \log_{1+\varepsilon} \frac{r_{\max}(v)}{r_{\min}(v)}) = O(\alpha|P|)$. *Further, this clustering can be done in* $O(n \log(n))$ *time.*

Proof. The proof of this theorem is based on the following lemma.

Lemma 4.2. *If* W_i *is the set of clusters that we add in phase* i*, then*

$$\sum_{v \in W_i} \log_{1+\varepsilon} \frac{r_{\max}(v)}{r_{\min}(v)} = O\left(\alpha \times \sum_{(p_1, p_2) \in E_{k_1} \cup E_{k_2} \cup \cdots \cup E_{k_i}} \frac{\mathrm{dist}(p_1, p_2)}{2^{k_i}}\right)$$

and the time taken in phase i *is* $O(|W_i| \log(n))$.

Using Lemma 4.2, we count the contribution of each edge of the λ-MST. Suppose, $e \in E_{k_i}$. Then, it does not contribute anything up to phase $i-1$. In any phase $j \geqslant i$, it contributes $\alpha\,\mathrm{len}(e)/2^{k_j}$. So, total contribution is at most $\alpha\,\mathrm{len}(e)\sum_{j=i}^{\infty}(1/2^{k_j}) \leqslant 2\alpha\,\mathrm{len}(e)/2^{k_i}$. Also, since $e \in E_{k_i}$, so $\mathrm{len}(e) \leqslant 2^{k_i+1}$. Therefore, the contribution of each edge is at most 4α and the total is $O(\alpha(\text{number of edges in } \lambda\text{-MST}))$, which is clearly $= O(\alpha|P|)$.

As for the total time taken, time taken in phase i is upper bounded by $|W_i| \log(n)$. Since, we add a total of $\sum_i |W_i| = n-1$ clusters, so total time taken is $O(n \log(n))$.

This completes the proof of Theorem 4.4. □

[2] Spans as in a spanning tree. A vertex set S spans a connected component in a graph $G = (V, E)$ if the graph $G' = (S, E \cap \binom{S}{2})$ is a connected graph.

4.6. Merging within phase i—Lemma 4.2

In this section we prove Lemma 4.2, using a series of intermediate results. We start by giving a graph theoretic result.

Lemma 4.3. *Given a tree T and k of its vertices $\{v_1, v_2, \ldots, v_k\}$ (possibly repeated), we can pair them up (if odd, 1 vertex would be left out) as $\{(v_{i_1}, v_{j_1}), \ldots, (v_{i_{k/2}}, v_{j_{k/2}})\}$, so that the set of paths from v_{i_l} to v_{j_l} are all edge disjoint in T.*

Furthermore, the tree T can be preprocessed in time $O(n \log n)$, such that once that is done, the pairing up of any given k vertices can be done in $O(k \log n)$ time.

Proof. We prove it by induction on the number of nodes in the tree. The base case is simple. In induction step, pick any leaf, say v of the tree T.

Case 1 *(v is not among the vertices to be paired).* Remove it from the tree and use the induction hypothesis to pair all the vertices.

Case 2 *(v is among the vertices to be paired).* Then v must be connected to some other vertex in T. Let it be u.

If u is also to be paired, then pair u and v and pair the rest of the vertices using induction hypothesis (hypothesis can be used after removing v and reversing the selection of u in T).

If u is not to be paired then invoke the induction hypothesis, with v removed and u selected. Then, every vertex would get paired up with some vertex in T. Suppose, u is paired with w. Then, pair v with w and retain the rest of the pairings. It is easy to see that all the paths are edge-disjoint.

Using the above strategy, we will design an algorithm to pair k vertices in $O(n)$ time. However, we need an algorithm which does this pairing in $O(k \log(n))$ time, with permitted preprocessing time of $O(n \log(n))$. Here is an overview of the construction—the details are not difficult to work out. As a preprocessing step, construct a balanced edge separator tree for the approximate MST. This can be done in $O(n \log n)$ time. Then to solve a pairing problem for k input points, we split the k points into two parts according to the side of the separator tree, each of them lies in. Then, we recursively pair the two parts, using the subtree they are associated with. Finally, two partial solutions are combined by pairing up unpaired vertex in the left half (if any) with an unpaired vertex in the right half (if any). The query time obtained is $O(k \log(n))$. □

We give some additional definitions.

Definition 4.10.

(1) G_i is defined to be a sub-graph of the λ-MST of the point set P with vertex set P and edge set $E_{k_1} \cup E_{k_2} \cup \cdots \cup E_{k_{i-1}}$. Note that it is a forest.
(2) C_i is defined to be the set of connected components of G_i.
(3) H_i is a forest over the connected components of G_i formed by edges in E_{k_i}. Its vertex set is C_i. The edge set is $\{(c_s, c_t) \mid \exists (p_s, p_t) \in E_{k_i},\ p_s \in c_s,\ p_t \in c_t\}$.

Observation 4.5.

$C_i =$ *The set of all the roots of the rooted forest* F_i.

This follows from the discussion in Section 4.5.

We shall use the Lemma 4.3 to *merge* the clusters in the form of what we call a *star*. The definition of a star follows.

Definition 4.11 *(Star).* A set of clusters $c_0, c_1, \ldots, c_k$ is called a star of the phase i with c_0 as center, if they satisfy the following conditions.

- c_0 spans a connected component in G_{i+1}.
- For each $j \in \{1, 2, \ldots, k\}$, $c_0 \cup c_j$ spans a connected component in G_{i+1}.

c_0 is called the center of the star and $c_1, c_2, \ldots, c_k$ are called the leaves of the star.

4.6.1. Merging of a star

For simplicity, we assume that number of leaves of the star is a power of two. First we make the following claim.

Lemma 4.4. *Given a star with center c_0 and leaves $c_1, c_2, \ldots, c_k$ we can pair each of $c_1, c_2, \ldots, c_k$ into $k/2$ pairs, say, $(c_1, c_2), \ldots, (c_{k-1}, c_k)$, such that*

$$\sum_{j=1}^{k/2} S(c_{2j-1} \cup c_{2j}) \leqslant \sum_{j=0}^{k} S(c_j) + k2^{k_i+1}.$$

Time taken for the pairing is $O(k \log(n))$.

Proof. From the definition of the star, $c_0 \cup c_k$ spans a connected component in G_{i+1}. So, there exists an edge in $\bigcup_{j=1}^{i+1} E_{k_j}$, which connects c_0 to c_k. Suppose, for each c_l, (v_l, v'_l) be the edge connecting c_0 to c_l with $v_l \in c_0$ and $v'_l \in c_l$.

Observation 4.6.

$$S(c_m \cup c_n) \leqslant S(c_m) + S(c_n) + \text{PathLength}\big(v'_m, v'_n\big),$$

where PathLength(v_m, v_n) *is the length of the path between v_m and v_n in the λ-MST.*

Observation 4.7.

$$\text{PathLength}\big(v'_m, v'_n\big) \leqslant \text{PathLength}(v_m, v_n) + 2 \times 2^{k_i}.$$

This follows from the fact that both the edges (v'_m, v_m) and (v'_n, v_n), which appear at the two ends of the path from v'_m to v'_n, are of length at most 2^{k_i}.

Since, c_0 spans a connected component in G_{i+1}, which is a sub-graph of the λ-MST, so c_0 spans a tree in G_{i+1}. Each of v_l is contained in c_0. Use Lemma 4.3 to pair each v_l, using the tree spanned by c_0 in G_{i+1}. Without losing generality, we assume that the pairing is $(v_1, v_2), (v_3, v_4), \ldots, (v_{k-1}, v_k)$. Then, since the paths are edge-disjoint, we get the following result:

$$\sum_{j=1}^{k/2} \text{PathLength}(v_{2j-1}, v_{2j}) \leqslant S(c_0).$$

Using Observations 4.6 and 4.7, we obtain

$$\sum_{j=1}^{k/2} S(c_{2j-1} \cup c_{2j}) \leqslant \sum_{j=1}^{k} S(c_j) + S(c_0) + k2^{k_i+1}.$$

Also, from Lemma 4.3, time taken in pairing v_i's is $O(k \log n)$. This is the time taken in pairing c_i's as well. This proves Lemma 4.4. □

Armed with Lemma 4.4, we now describe the procedure to merge a star. We use the following procedure to merge the leaves of the star.

Merge-Star($\{c_0, c_1, c_2, \ldots, c_k\}$). The procedure works in iterations. In each iteration, we try to merge a star. At the end of each iteration, we are left with a star with lesser (roughly half) number of leaves than we had at the beginning of the iteration. The center of the stars in each iteration remains c_0.

Let $s_1, s_2, \ldots, s_{k'}$ be the leaves of the star to be merged in the current iteration. Here k' is initialized with k, and s_i's are initialized with c_i's.

- If $(k' = 1)$, then merge the only leaf with the center of the star and exit.
- Use Lemma 4.4 to pair up the leaves of the star. Without loss of generality, the pairing is $(s_1, s_2), \ldots, (s_{k'-1}, s'_k)$.
- Merge the clusters in a pair.
- Repeat the procedure with

  $$k' \leftarrow k'/2$$

  and

  $$\{s_1, s_2, \ldots, \} \leftarrow \{s_1 \cup s_2, s_3 \cup s_4, \ldots\}.$$

  (Note that this is again a star.)

Lemma 4.5. *A star of phase i with c_0 as center and $c_1, c_2, \ldots, c_k$ as leaves can be merged by adding a set of clusters W, such that*

$$\sum_{v \in W} \log \frac{S(v)}{2^{k_i}} = O\left(k + k \log\left(\frac{S(c_0)}{k2^{k_i}} + \sum_{j=1}^{k} \frac{S(c_j)}{k2^{k_i}}\right)\right).$$

Also, the time taken in the merging of the star is $O(|W| \log n)$.

The proof is described in Appendix A.

4.6.2. Merging of a connected component of H_i

Now that we know how to merge a star, we describe how to merge a connected component of H_i. Let that component be C. Let $D = \bigcup_{c \in C} c$ where c is a star. In the following procedure to merge the components, A denotes the set of clusters yet to be merged. It is initially assigned the value C.

Merge-Component.

- If $|A| = 1$, then there is nothing to be done.
- Define T' to be a tree, with vertex set A and edge set as $\{(c_1, c_2) \mid c_1, c_2 \in A \wedge \exists (u_1, u_2) \in G_{i+1},\ u_1 \in c_1, u_2 \in c_2\}$. (Note that this graph is always a tree.)
- Find a minimum vertex cover $\mathcal{V}$ of T'. Note that the smallest vertex cover of a tree has size at most half the size of the tree $|\mathcal{V}| \leqslant \frac{1}{2}|A|$.
- For each cluster $c_0 \in \mathcal{V}$, we have a set of clusters $\{c_1, c_2, \ldots, c_k\} \subset A \backslash \mathcal{V}$, directly connected to it (in other words *covered by it*).

Observation 4.8. *$c_0, c_1, \ldots, c_k$ form a star with c_0 as center and $c_1, c_2, \ldots, c_k$ as leaves.*

Use the vertex cover $\mathcal{V}$ to *partition* A into *stars*. Since, each star is centered around a vertex of the vertex cover, the number of stars is $|\mathcal{V}|$.

- Merge the stars using the procedure referred to in Lemma 4.5. After the merging, number of components left is $|\mathcal{V}| \leqslant |A|/2$. So, we have reduced the number of components from $|A|$ to at most $|A|/2$.
- Repeat the procedure with A replaced by set of merged stars.

Lemma 4.6. *It is possible to merge a connected component C of H_i by adding a set of clusters W, such that*

$$\sum_{v \in W} \log_{1+\varepsilon} \frac{r_{\max}(v)}{r_{\min}(v)} = O\left(\alpha \times \frac{S(\bigcup_{c \in C} c)}{2^{k_i}}\right).$$

Time taken is $O(|W| \log(n))$.

The proof is described in Appendix A.

The proof of Lemma 4.2 follows by using the bound of Lemma 4.6 on each of the connected components of H_i and then adding the contributions. This completes the proof of Lemma 4.2. □

Appendix A

Proof of Lemma 4.5. Since in each iteration of the *merge-star* procedure, the number of leafs decreases by a factor of two, so in the iteration t, $k' = \frac{k}{2^t}$.

Observation A.1. *In the tth iteration,*

$$k' = \frac{k}{2^t}.$$

Also from the Lemma 4.4, in each iteration, the sum of the $r_{\max}$ values of clusters increase by at most $S(c_0) + k2^{k_i+1}$. So, we get the following observation.

Observation A.2. *In the tth iteration,*

$$\sum_{j=1}^{k'} S(s_j) \leqslant \sum_{j=1}^{k} S(c_j) + t\big(S(c_0) + k2^{k_i+1}\big).$$

Observation A.3. *Time taken in the tth iteration is $O(\frac{k}{2^t}\log(n))$. This follows, from the fact that, in the merging algorithm, pairing up of the clusters takes $O(k'\log(n))$ time and merging takes $O(k')$ time. Since, in the tth iteration, $k' \leqslant 2k/2^t$, so the time taken is $O(\frac{k}{2^t}\log(n))$.*

Now, the *contribution* of set of clusters that are added in the tth iteration is given by

$$\sum_{j=1}^{k'/2} \log \frac{S(s_{2j-1} \cup s_{2j})}{2^{k_i}}.$$

Using weighted Geometric mean $\leqslant$ weighted arithmetic mean,

$$\sum_{j=1}^{k'/2} \log \frac{S(s_{2j-1} \cup s_{2j})}{2^{k_i}} \leqslant \frac{1}{2}k' \log\left(\sum_{j=1}^{k'/2} \frac{S\big(s_{2j-1} \cup s_{2j}\big)}{k'2^{k_i}}\right).$$

Using Observations A.1 and A.2, we get for the tth iteration

$$\sum_{j=1}^{k'/2} \log \frac{S(s_{2j-1} \cup s_{2j})}{2^{k_i}} \leqslant \frac{k}{2^{t-1}} \log\left(2^t \frac{\sum_{j=1}^{k} S(c_j) + tS(c_0) + t2^{k_i+1}}{2^{k_i}k}\right).$$

So the total contribution is

$$\sum_t \frac{k}{2^{t-1}} \log\left(2^t \frac{\sum_{j=1}^{k} S(c_j) + tS(c_0) + t2^{k_i+1}}{2^{k_i}k}\right).$$

It is easily shown that this is

$$O\left(k + k\log\left(\frac{S(c_0)}{k2^{k_i}} + \sum_{j=1}^{k} \frac{S(c_j)}{k2^{k_i}}\right)\right).$$

Also, adding time taken in each iteration from Observation A.3, we find the total time taken to be $O(k\log n)$. This proves the Lemma 4.5. □

Proof of Lemma 4.6. Consider any iteration of the procedure. Suppose, c_{rs} is the sth cluster of the rth star of A. Suppose, h_r is the number of clusters in the rth star.

Then, from Observations 4.4 and 4.3 and Lemma 4.5, we get—we can merge all the stars created in this iteration by adding a set of clusters W, with

$$\sum_{v\in W} \log_{1+\varepsilon} \frac{r_{\max}(v)}{r_{\min}(v)} = O\left(\alpha \sum_{v\in W} \log \frac{S(v)}{2^{k_i}}\right) = O\left(\alpha \sum_r \left(h_r + h_r \log\left(\sum_{s=0}^{h_r} \frac{S(c_{rs})}{h_r 2^{k_i}}\right)\right)\right).$$

From (geometric mean $\leqslant$ arithmetic mean), we get

$$\sum_{v\in W} \log_{1+\varepsilon} \frac{r_{\max}(v)}{r_{\min}(v)} = O\left(\alpha\left(\sum_r h_r + \left(\sum_r h_r\right) \log\left(\frac{\sum_{rs} S(c_{rs})}{2^{k_i} \sum_r h_r}\right)\right)\right).$$

Clearly, $\sum_{rs} S(c_{rs}) \leqslant S(D)$, and $\sum_r h_r = |A|$. So, this gives

$$\sum_{v\in W} \log_{1+\varepsilon} \frac{r_{\max}(v)}{r_{\min}(v)} = O\left(\alpha\left(|A| + |A| \log\left(\frac{S(D)}{2^{k_i}|A|}\right)\right)\right).$$

Now, in each iteration $|A|$ reduces by at least a factor of two. Initially, it is $|C|$. So, the total contribution from all the clusters added in all the iterations is

$$O\left(\alpha|C|\left(1+\frac{1}{2}+\frac{1}{4}+\cdots\right)\left(1+\log\left(\frac{S(D)}{2^{k_i}|C|}\right)\right) + \alpha|C|\left(1+\frac{\log 2}{2}+\frac{\log 4}{4}+\cdots\right)\right)$$
$$= O\left(\alpha\left(|C| + |C| \log\left(\frac{S(D)}{2^{k_i}|C|}\right)\right)\right).$$

Now, in the sub-graph spanned by D, there are $|C|-1$ edges of length between 2^{k_i} and 2^{k_i+1}. So, from definition of $S(D)$, $|C| \leqslant \frac{S(D)}{2^{k_i}}$. Also, $|C| \log(\frac{S(D)}{2^{k_i}|C|})$ is $O(\frac{S(D)}{2^{k_i}})$. So, total contribution is $O(\alpha S(D)/2^{k_i})$.

The time taken in kth iteration is easily seen to be $|A| \log(n)$ (using Lemma 4.5). Since, in the kth iteration, $|A| \leqslant |C|/2^k$, so total time taken in all the iterations is $O(|C| \log(n) + |C| \log(n)/2 + |C| \log(n)/4 + \cdots) = O(|C| \log(n))$. This proves Lemma 4.6. □

Appendix B. Constructing an approximate MST

Har-Peled described two algorithms for computing approximate MSTs. One takes $O_d(n \log(n) + \frac{n}{\delta^d})$ time to construct a $1+\delta$-MST. It uses the Callahan Kossaraju's Well Separated Pair Decomposition. He also gives an algorithm which takes $O(n \log(n))$ time to construct an nd-MST. We can use the first algorithm to construct a 2-MST for our purposes.

Alternatively, we can construct a λ-MST using the techniques of this paper, without using WSPD. The procedure is as follows.

(1) Construct nd-MST T_1 of P using the technique of Har-Peled. Note that this does not use WSPD.
(2) Using techniques of this paper and T_1, construct an approximate Voronoi Diagram V with approximation of $\varepsilon = \mu$. This would take $O(n \log n)$ space.
(3) Given a Voronoi Diagram (or approximate Voronoi Diagram), we define its Delaunay graph as a graph over sites of the diagram, with edges between points p_1 and p_2 if and only if cells of p_1 and p_2 share a boundary. Construct the Delaunay graph G of V.
(4) The MST of G is an $1+3\mu$-MST of the point set P.

The proof is based on the fact that approximate Delaunay graph contains an approximate MST. This can be contrasted with a similar result in the case of exact MST and Delaunay graph. We shall prove it by proving that given any cut (P_1, P_2) in P, there exists an edge (p'_1, p'_2) in the Delaunay graph such that $\text{dist}(p'_1, p'_2)$ is at-most $(1+3\mu)\,\text{dist}(P_1, P_2)$.

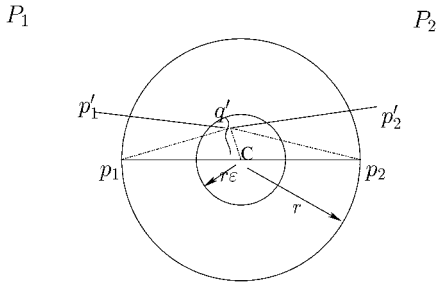

Fig. 5. A cut in an approximate MST.

Proof. Suppose, p_1, p_2 be the points such that $p_i \in P_i$ and $\text{dist}(p_1, p_2) = \text{dist}(P_1, P_2)$. Consider the ball $b(C, r)$ with $\overline{p_1, p_2}$ as diameter (Fig. 5). It is easy to show that no point of P is contained in $b(C, r)$. Consider all the Voronoi cells that intersect the ball $b(C, \varepsilon r)$. Observe that those cells cannot all correspond to one of P_i. So, there is a point (say q') in $b(C, \varepsilon r)$ which lies at the boundary of two Voronoi cells, one of them corresponding to a point in P_1 and other to a point in P_2. Let these points be p'_1 and p'_2. So, both p'_1 and p'_2 are ε-nearest neighbor's of q'. So, $\text{dist}(q', p'_i) \leqslant (1 + \varepsilon)\,\text{dist}(q', p_i)$. Using this and from geometry, we get $\text{dist}(p'_1, p'_2) \leqslant \text{dist}(p'_1, q') + \text{dist}(p'_2, q') \leqslant (1 + \varepsilon)(\text{dist}(p_1, q') + \text{dist}(p_2, q')) \leqslant (1 + \varepsilon)(r + r\varepsilon + r + r\varepsilon) \leqslant (1 + \varepsilon)^2\,\text{dist}(P_1, P_2) \leqslant (1 + 3\varepsilon)\,\text{dist}(P_1, P_2)$. Since there is an edge in the Delaunay graph between p'_1 and p'_2, the result follows. □

Appendix C. Hierarchical clustering for construction of PLSBs

Our techniques for the construction of PLSBs, like that of Har-Peled [11], are based on a hierarchical clustering of the point set. These methods primarily differ in the way the hierarchical clustering is constructed, however the underlying ideas are similar. We start with the individual points forming the leaves of the hierarchical clustering. In each step two clusters are merged on the basis of some measure of "proximity" between the clusters. A set of balls is constructed around a representative from each of these clusters. These balls are used to determine which cluster contains a point that is approximately closer to the query point. This is determined by the representative around which the ball containing the query point is constructed. When the query point is sufficiently far from both the clusters, then a point from either cluster can be reported and therefore there is no need to construct more balls to distinguish between the proximity of the query point from the points belonging to these clusters. This process is repeated until all the points are merged into a single cluster.

The set of all the balls constructed in this manner constitute the balls of the PLSB problem.

C.1. Answering PLSB using compressed quadtree

Har-Peled [11] described an efficient data structure for answering a PLSB problem by first approximating the balls with axis-parallel cubes extracted from a carefully constructed hierarchical grid and then constructing a quadtree search tree over these cubes. We give below an outline of the data structure.

Consider a set $\mathcal{B}$ of m balls, constituting a PLSB problem. We wish to construct a data structure over $\mathcal{B}$, that given a query point q, reports the smallest ball containing q efficiently. This is done by approximating the balls by a set of cubes from a hierarchical grid (Fig. 6).

For a real number u, let $\mathcal{G}(u)$ be the partition of $\Re^d$ into a uniform axis parallel grid centered at the origin, where the side-length of the grid is $2^{\lfloor \log u \rfloor}$. By varying the value of u, we get a multi-resolution grid that covers space.

Now consider a ball $b = B(p, r)$ centered at p of radius r. Let $GC(b, \varepsilon)$ be the set of cubes of the grid $\mathcal{G}(r/\varepsilon d)$ that intersects b. Let $b_\varepsilon = \bigcup_{c \in GC(b,\varepsilon)} c$. Then,

- b_ε is an ε-approximation to b; and
- $|GC(b, \varepsilon)| = O(1/\varepsilon^d)$.

Now consider $\mathcal{B}$. Using the technique described above, one can derive a set $GC(\mathcal{B}, \varepsilon) = \bigcup_{b \in \mathcal{B}} GC(b, \varepsilon)$ of $O(m/\varepsilon^d)$ cubes from the axis-parallel hierarchical grid that ε-approximates $\mathcal{B}$. There is also one "infinite" cube,

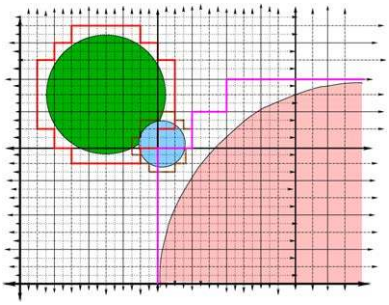

Fig. 6. ε-PLSB data structure.

that covers the whole space. This is a background cube that serves for answering ε approximate nearest neighbor queries when the query point is so far from P, that any point $x \in P$ is an ε-nearest neighbor to the query point.

Note that wherever there is an overlap of cubes from different balls, priority is always given to the smaller cube, i.e., the cube corresponding to the smaller ball.

Let $\mathcal{C}$ be the set of all the cubes generated. Since all the cubes are taken from a hierarchical grid representation of $\Re^d$, these cubes can be stored in a quadtree, $\mathcal{Q}$, and the quadtree can be used to answer point-location queries among the prioritized cubes. The compressed quadtree contains regions that are either axis parallel cubes or the annulus (difference) of two cubes, one contained inside another. The regions are disjoint and provide a covering of space. Furthermore, each region is associated with the ball it corresponds to. In case of overlap, the region is associated with the smallest ball. Using standard techniques [1], $\mathcal{Q}$ can be computed in time $O(|\mathcal{C}| \log |\mathcal{C}|)$. The leaves of $\mathcal{Q}$ can then be preprocessed for point-location using the data structure of [10]. A point location query for a point q can then be answered in $O(\log |Q|) = O(\log (m/\varepsilon))$ time.

References

[1] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A.Y. Wu, An optimal algorithm for approximate nearest neighbor searching in fixed dimensions, J. ACM 45 (6) (1998) 891–923.

[2] Sunil Arya, Theocharis Malamatos, Linear-size approximate Voronoi diagrams, in: Proc. of SODA, 2002.

[3] Sunil Arya, Theocharis Malamatos, David Mount, Space-efficient approximate Voronoi diagrams, in: Proc. of ACM STOC, 2002.

[4] Sunil Arya, David M. Mount, Approximate nearest neighbor queries in fixed dimension, in: Proc. of SIAM–ACM SODA, 1993, pp. 271–280.

[5] D. Attali, 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.

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

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

[8] Kenneth L. Clarkson, An algorithm for approximate closest-point queries, in: Proc. 10th Annu. ACM Sympos. Comput. Geom., 1994, pp. 160–164.

[9] J. Erickson, Nice points sets can have nasty Delaunay triangulations, in: Proc. 17th Annu. ACM Sympos. Comput. Geom., 2001.

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

[11] S. Har-Peled, A replacement for Voronoi diagrams of near linear size, in: Proc. of IEEE FOCS, 2001, pp. 94–103, full version available from http://www.uiuc.edu/~sariel/papers.

[12] P. Indyk, R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, in: Proc. 30th Annu. ACM Sympos. Theory Comput., 1998, pp. 604–613.

[13] Piotr Indyk, PhD thesis, Stanford University, 1999.

[14] Eyal Kushilevitz, Rafail Ostrovsky, Yuval Rabani, Efficient search for approximate nearest neighbour search in high dimensional spaces, in: ACM Symposium on Theory of Computing, 1998, pp. 614–623.