IJITCE

ISSN 2347- 3657

International Journal of Information Technology & Computer Engineering

Email : ijitce.editor@gmail.com or editor@ijitce.com

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 11, Issue 1, 2023

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

Ms. R. Latha Priyadharshini, Mr. S. Satheesh, Mrs. M. Indra Priya, Ms. R. Roopa
Associate Professor [3] Assistant Professor [1,2,4]

lathapriyadharshini.r@actechnology.in, ssatheesh@actechnology.in, indrapriya.m@actechnology.in, roopa.r@actechnology.in

Department of CS & BS, Arjun College of Technology, Thamaraikulam, Coimbatore-Pollachi Highway, Coimbatore, Tamilnadu-642 120

Abstract

Our proposed solutions to the nearest neighbour searching problem for Point Location in Balls are an alternative to Sariel Har-Peled's recent work on Approximate Voronoi Diagrams, which maintains the logarithmic search time, and reduces the space bound to linear. This work is published in the Proc. of IEEE FOCS in 2001 and can be accessed online at http://www.uiuc.edu/~sariel/papers. We achieve this by streamlining the building of the algorithm that reduces the number of balls generated by it to O(n log n), as described in [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]. In order to accomplish linear space decomposition for closest neighbour searches, we further decrease the ball count by introducing a novel hierarchical decomposition strategy and expanding upon PLEBs. Our data structures are constructed with a temporal complexity of O(n log n).

Keywords: Voronoi diagrams; Approximate nearest neighbor; Data structures

1. Introduction

I will use a metric space and a collection of n points as P. Constructing a P-based data structure that effectively permits closest neighbour searches is one of the most basic challenges with varied applications. This issue has workable answers in two-dimensional planar space, but it becomes non-trivial in three-dimensional and higher-dimensional Euclidean spaces [5,9]. An overarching strategy for nearest neighbour searching is to construct a Voronoi diagram of P (Hereinafter Vor(P)), a space partition that includes all points that are closer to a given point of P than any other point in P. Each cell in the diagram contains one such point. Computer scientists in computational geometry rely on voronoi diagrams for a wide variety of tasks, such as surface reconstruction, learning, motion planning, clustering, and more. Voronoi diagram generalisations to objects other than point sets are also very helpful (for a comprehensive review, see Aurenhammer [6]). Despite having many uses, voronoi diagrams have a major flaw: they are very structurally complicated.

computational bottleneck and the resulting ity. The exponentially complex worst-case complexity in Rd is known to be θ (n ⌈d/2) with constants in d. As a result, it becomes computationally impossible to generate and store these diagrams as their dimensions increase.

Therefore, in recent years, researchers have focused on finding alternative solutions to closest neighbour searches in greater dimensions [1,4,8,12–14]. The related relaxed issue of approximate nearest neighbour search has gained prominence and has showed promise of practical solutions because the exact problem looks closely tied to the complexity of Voronoi diagrams. We want to get a data structure by preprocessing the dataset P, given a constant ε < 1. D,

that, given a query point q, efficiently returns a point P such that the distance between p and q, denoted as pr P, is less than or equal to (1 ⊑ ε) times the distance between pr and q. P is an ε closest neighbour (ε-NN) of q, and dist might be any arbitrary measure in this case.

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 11, Issue 1, 2023

The work of Indyk and Motwani [12], where they simplified the issue of ε-NNS to Approximate Point Location in Equal Balls (ε-PLEB), is particularly relevant to this research.

Definition 1.1. When given a query point q and a parameter r, an ε-PLEB(P, r) data structure finds a point p in P such that dist(p, q) < r, if one exists, given P and r. Returns nil if there is no point pr ∈ P such that dist(pr,q) < r(1 + ε). Anything else may result in a return. It is referred to as PLEB(P, r) when ε equals zero.
A new ring-cover tree was used by Indyk and Motwani to decrease ε-NNS to ε-PLEB. A basic hash-based method is used to resolve the ε-PLEB. Nevertheless, their reduction was extremely complicated; however, Har-Peled [11] made a complete improvement by presenting an alternate space decomposition known as an Approximate Voronoi Diagram (AVD), which allows for approximate nearest neighbour searching and has a much lower space complexity. His findings are summarised below.

Theorem 1.1 *(Har-Peled). Given P of n points in R^d and a parameter $\varepsilon > 0$ one can compute a set $C(P)$ 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 $C(P)$ 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

If AVD could be made linear-sized instead of near linear, it would be a significant improvement above Har-Peled's previous work. In this study, we generalise the concept of PLEBs to Point Location in Smallest Ball (PLSB), where the balls may have variable radii, and we accomplish space reduction by extending some of the principles of Har-Peled. For more specific definitions, see Section 4, however we also provide an approximation of it. This issue has also been implicitly—and more ad hocly—addressed by Har Peled [11].
It should be noted that PLEB is an example of PLSB where the radii of each ball is equal.
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.*

Appendix C provides a synopsis of the approach, which is comparable to Har-Peled [11]. We shall limit the number of balls in the data structure, which is directly proportional to the space complexity, in the remaining work.
In the part that follows, we generate a set of balls straight from the clustering and decrease their number by a factor of log n, thereby eliminating the recursive nature of the Har-Peled's approach. There is no change to the logarithmic search time and an improvement of O(log n) to the preprocessing time.

on Section 4, we use a novel hierarchical clustering approach, which is also grounded on the MST of P, to enhance the space bound to linearity. Using a quicker approximation MST approach that requires O(n log n) time—the same as Har-Peled—we are able to minimise the preprocessing time, which is practically quadratic due to the precise MST construction. As a general rule, in Har-Peled's data structure, we combined two clusters whenever their MST edge was the smallest. A comparison of hierarchical

This clustering method differs from Har-Peled's in that, rather than attempting to combine the two closest clusters, we merge many clusters in stages such that the resulting new cluster has a small diameter (Section 4).

Using Callahan and Kosaraju's [7] well separated space decomposition, Arya and Malamatos [2] independently obtained a comparable result. Additionally, they achieve space-time trade-offs for generalising the Voronoi cells with multiple

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

representatives, an improvement that has been made by Arya et al. [3]. A potential

The number of balls produced by our technique is much lower—O(n)—than that of

A other search structure, not the BBD tree, could be able to take use of this, and the time complexity would be O((α√d) n).

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.,* $\frac{|QA|}{|QB|} < 1 + \epsilon$.

In addition, a clustering approach for organising the points into a hierarchical tree is suggested by Har-Peled [11]. If you were to grow balls around all the spots, you would end up with linked components. Further, we use the linked parts to build a forest, also known as a cluster tree.

At the outset, we work with only the collection of points. Accordingly, we begin with n linked components that comprise the individual points. There are n trees in the matching forest, and each tree has one leaf that represents a point. The balls are then evenly grown around all the spots. At the spots where two components' balls touch, a new component is formed by merging the two components, and this new component takes the place of the two components at those locations. By randomly hanging one tree on top of the other, we combine the two trees in the matching forest that represent the merging components (see Fig. 1).

Now we additionally attach a value of rloss(p) to the tree's root, p, from the other tree. When these parts come together, this number represents the ball's radius. It is half of the distance between the two components' nearest locations, in fact.

Thus $r_{loss}(p)$ = 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.

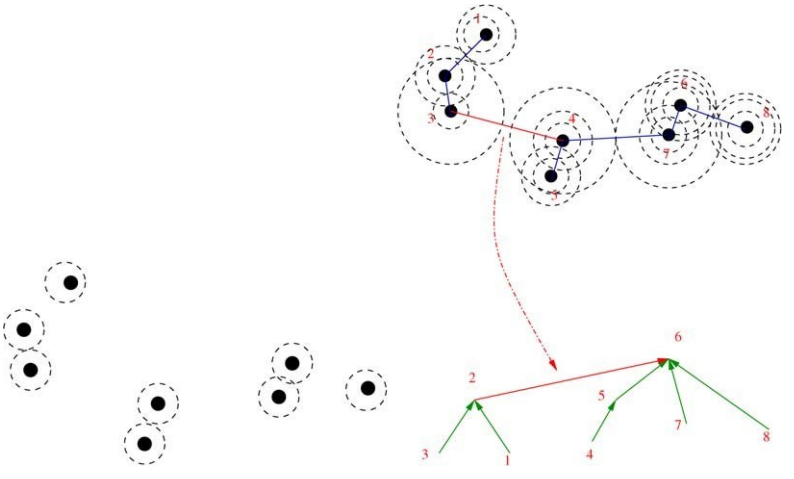

Fig. 1. Constructing the cluster tree.

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 11, Issue 1, 2023

- 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 $L_{\max}.|P|$.

Any subtree (cluster) that forms during building also has the aforementioned qualities.
It requires O(n2) time to calculate an accurate d-dimensional MST for high d, but Har-Peled settles for a nd-factor approximation that can be done in O(n log n) time.
In order to answer the estimated closest neighbour queries in O(log(n)) time, he uses the nd-MST to build a nd-stretch Hierarchical Clustering and then provides a technique for building a family of PLEBs (Point Location in Equal Balls). The search continues by iteratively scanning the remaining points after trimming certain clusters based on some PLEBs. We need to build O(log n) PLEBs for every recursive call, and there are O(log n) levels total. At last, a compressed quadtree [1] is created by merging all of the O(n log2 n) balls. The space limit is about O(n log2 n) and is dependent on the number of balls created by the PLEB family.

Improving the space bound

2.2. Here we provide an alternative viewpoint, explaining how we might reduce the amount of balls needed to answer approximate closest neighbour queries by removing overlapping and unnecessary balls from the collection of balls centred around a point p P. The approximate closest neighbour question may be answered by providing the centre of the smallest ball that includes a query point, and we provide an independent approach for building balls around P points.

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, len*(e)* $\leq d(P_1, P_2) \leq \lambda$ 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*For any point p in a directed tree T, there exists an out-edge from p to its parent with a value rloss(p) that satisfies the following property:

If we have C1 and C2 as connected components obtained from building balls of radius r around all points and F as the forest obtained by removing all edges larger than r from T and X as the partition of P into subsets induced by the connected components of the forest, then C1 X C2, where X Y, and A and B are in the same component in X if and only if they are in the same component in Y.

The following is the process for constructing the nd-stretch hierarchical clustering using the nd-MST:

We arrange the nd-edges MST's in ascending order of their len(.) values. Afterwards, we go down the list. By inserting an edge between the roots of the two subtrees, we are able to link the two subtrees to which x and y belong in each iteration. We do this by taking the next edge, e xy, from the sorted list and hanging the smaller tree onto the bigger one. With this edge e, we connect the value rloss(e) = len(xy)/2.

At the very beginning of the tree, at the root t, we set rloss(t) = max(rloss(pi)), where pi 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* $\log n$. (*Hangingthe 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.

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 11, Issue 1, 2023

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

Definition 3.4 (parent), parenti(p). In the tree that is created by the λ-stretch hierarchical clustering of P, parent(p) is defined as the parent of node p. The ith parent of node p in the tree that is generated by the λ-stretch hierarchical clustering of P is defined as parenti(p), where parent0(p) is p and parenti(p) is parent(parent(i−1)(p)). The function parenti(p) is defined up to the point when it becomes the root of the tree that is generated by the λ-stretch hierarchical clustering of P, but it is not defined after the root.

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

2.3. *Construction of balls (algorithm* ConstructBalls$(P, \gamma))$

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., $r_{loss}(p_1) \leq r_{loss}(p_2) \leq \leq r_{loss}(p_m)$. 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 $r_i = r_{loss}(p)(1 + \gamma/3)(j^{-1})$ for $j = 0, \ldots, M - 1$, where $M = \lceil \log_{(1+\gamma/3)}(1+\gamma/3)(r_{high}(p)/ r_{loss}(p)) \rceil$.

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

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

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

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.

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

Find the biggest ball such that for every query point beyond it we may claim another point exists which is a close neighbour of the query point with a limited cumulative approximation error; this will allow us to restrict the number of balls we build around a particular point.

If the query point is not within the biggest ball that can be created around the point x, then parent(x) is a neighbour of the query point with some accumulated approximation error, as shown in the following lemma, which also states that rhigh(x) is a limit on the radius of the largest ball. Therefore, we may disregard the point x, but this will result in an approximation mistake that will add up over time.

The cumulative mistake is a factor of 1 γ/(3 log n), according to the rhigh(x) formulation. As we shall see in a bit, this number is selected in such a way that the cumulative approximation mistakes can only grow up to a reasonable limit.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

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], $\|zx\| \le 2n\lambda r_{\text{loss}}(x) = (\gamma/(3\log n))r_{\text{death}}(x) < (\gamma/(3\log n))\|qx\|$. Therefore, $\|zq\| \le \|qx\| + \|zx\| \le (1+\gamma/(3\log n))\|qx\| \le (1+\gamma/(3\log n))(1+\alpha)d_P(q)$.

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

If the query point is not in the biggest ball surrounding the current candidate, we may recursively consider the parent of the current candidate as the next closest neighbour candidate. This is shown in the following lemma. Any time we move the candidate closer to the parent, our guess becomes more inaccurate.

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 $r > r_{\text{high}}(\text{parent}^i(p))$ $\forall 0 \le i \le j-1$, 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. □

Since we are making balls such that each one is larger by a factor of 1/3 compared to the one before it, we argue that reporting the nearest neighbour might result in an additional approximation error factor of 1/3 for the following lemma. What this means is that even though p is closer, we may end up reporting a different point x instead of p if q is in one of the balls in the ballset of p and p is the closest neighbour of q. This is due to the fact that the ball with q centred at x is smaller than the ball with q centred at p, even though p is closer. We can do this by using a discrete collection of balls, but there is an extra approximation error factor of $1+\gamma/3$, which means that we will report 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 $r_{\text{loss}}(t) \le r \le r_{\text{high}}(t)$ 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 $x \ne p$. 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 $d_P(q) \le \|xq\| \le (1+\gamma/3)(1+\alpha)d_P(q)$.

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

In this case, $\|pq\| > r_{\text{high}}(t) \ge r > \|xq\|$. 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 $\le r_{\text{low}}(t)$ and $r \ge r_{\text{loss}}(t) > r_{\text{low}}(t)$ 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 $r_j < \|pq\| \le r_{j+1} = (1+\gamma/3)r_j$.

Therefore, $|xq| \le r \le r_{j+1} = (1+\gamma/3)r_j < (1+\gamma/3)|pq| \le (1+\gamma/3)(1+\alpha)d_P(q)$. This means that x is a near-NN of q in P that is $(1+\gamma/3)(1+\alpha)$-approximate.

Assume that the query point q is included in the ball with radius rloss(p) for every given location p. Then, it's clear that the query point q's nearest neighbours can only be points in the cluster formed by the subtree of the hierarchical clustering tree rooted at p. Even then, we have no idea which of these stances is nearest; p may be one of them. This can only be accomplished by creating additional balls around p with radii that are compatible with the ballsets.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

comprised of nodes that are children of p. This method has the potential to reveal which of p's progeny is geographically nearest to the point q that we're trying to identify. There is no difference between this and the set of balls generated by algorithm ConstructBalls in accordance with rule 2. It is worth noting that the child nodes of p would be more appropriate representations of the children farther down the hierarchy than the balls that may be built around p.

If we use method ConstructBalls to produce balls, the previous lemma shows that in that case, we can only answer the queries with a further compounded approximation inaccuracy of a factor of $1 + \gamma/3$.

Lemma 3.4. *For any query point q, if p is a (1 α)-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 $r < r_{loss}(p)$ and iscentered at x, then x is a $(1 + \gamma/3)(1 + \alpha)$-approximate NN of q in P.*

Proof. If $q \not\in B(p, r_{loss}(p))$, then $\|pq\| > r_{loss}(p) > r \geq \|xq\|$ implying that x is closer to q than p and therefore trivially $\|xq\| \leq (1 + \gamma/3)(1 + \alpha)d_P(q)$.

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

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

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

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

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

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

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

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

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

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 $\gamma = \varepsilon/2$, be of radius r, centered at x, then x is a $(1 + \varepsilon)$-approximate NN of q in P.*

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

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

– $r_{loss}(p) \leq r \leq r_{high}(p)$:

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

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

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

Case II. $r > r_{high}(\text{parent}^{i}(p))$ *and* $r < r_{high}(\text{parent}^{(i+1)}(p))$ *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 $r_{loss}(\text{parent}^{i+1}(p)) \le r \le r_{high}(\text{parent}^{i+1}(p))$, then by the construction in algorithm ConstructBalls, since there exists a ballset in $[r_{low}(\text{parent}^{i+1}(p)), r_{high}(\text{parent}^{i+1}(p))]$, 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 $r < r_{loss}(\text{parent}^{i+1}(p))$, then using Lemma 3.4, x is a $((1+\gamma/(3\log n))^{i+1}(1+\gamma/3)$-NN of q in P.
Also $i+1 \le \log n$, 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. $r > r_{high}(t)$, *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 $> r_{high}(p)$. 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 $i \le \log n$, 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 $(1+\gamma/3)(1+\gamma/(3\log n))^{\log n} \le 1+2\gamma$, x is a $(1+2\gamma)$-approximate NN of q in P.
Hence x is a $(1+\epsilon)$-approximate NN of q in P. □

2.5. 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.

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

As we showed in the last section, the recursive nature of the data structure proposed by Har-Peled [11] may be eliminated, resulting in an O(n log(n)) space. We generate balls of radius between rloss(p), rloss(p) (1), rloss(p) (1 ″)2,..., rdeath(p) around each point in the database, which causes the additional log(n) factor to emerge. Every point in the database contributes Ω(log(n)) since rdeath(p) is defined as Ω(n/ε) times rloss(p). We really do not need rdeath(p) to be so high, however. It is necessary for rdeath to be the distance beyond which no point in the subtree(p) is any closer to the query point q than any other point in the tree. Assuming that diam is the diameter of a collection of points, an acceptable distance for that may be O(diam(subtree(parent(p)))/ε.
Despite using this rdeath definition, the number of balls that might be created could still be O(n log(n)). Take the following example to illustrate the point: there is a line that connects n locations (let's call them p1, p2,..., pn) such that

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

the distance between points pi and pi+1 is 1 plus ih, where h is

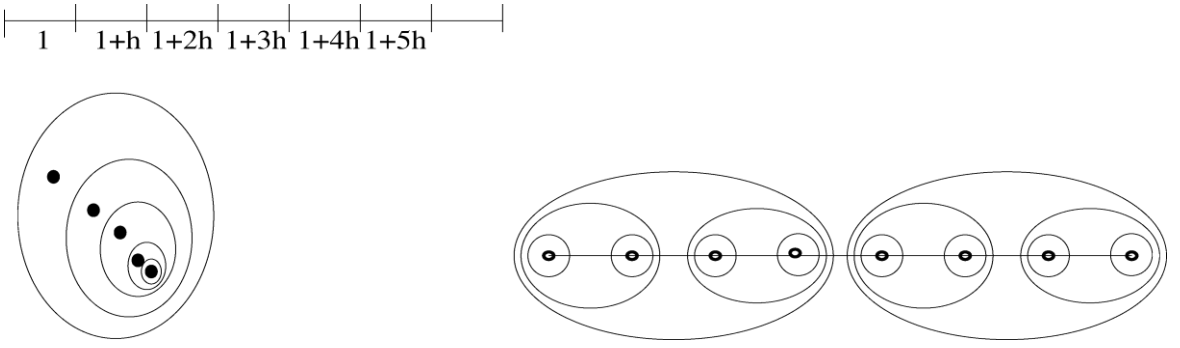

Fig. 2. Two different clustering for almost equidistant points in a line.

4 so little that it defies calculation. The MST is obviously only one chain. Even if we rethink rdeath as proposed, the quantity of balls produced by the method in [11] (and its adaptation here) remains Ω(n log(n)). The issue is that while building the hierarchical clustering, the method in [11] doesn't take the edge-lengths of the approximation MST into account, other from their relative ordering. We rethink the way hierarchical clustering is built in order to address that issue. We also care about the absolute values of the MST's edge lengths, not only their relative ordering; in other words, we don't differentiate between edges with almost similar lengths. Hence, the hierarchical clustering in the first case may seem different, as shown in Figure 2. Obviously, the second one is better; we'll go back to this specific instance 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^r 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^r such that (i) p is a *PLSB* solution for q^r *and* (ii) q^r 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 $(p^r, r^r) \in Q$, such that $r^r \leq r$ and there exists a point $q^r \in b(p^r, r^r)$ such that $\text{dist}(q, q^r) \leq r^r\varepsilon$ and $b(p^r, r^r)$ is the smallest ball containing q^r among balls from $\{b(p, r) \mid (p, r) \in Q\}$ (p^r is solution to PLSB query for q^r).
   (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).

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

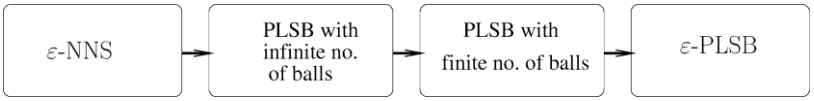

Fig. 3. The reduction of ε-NNS to ε-PLSB—the intermediate steps illustrate the proof strategy.

If we have a collection of points P, we can build a special set of balls B around those points (see to Definition 4.5) in such a way that the value of ε-PLSB in B gives us the approximate neighbour in P. To do this, we use the method shown in Figure 3.
A meticulously built hierarchical clustering of the point set P accomplishes the first reduction phase. Quite broadly, our hierarchical clustering is defined. The leaves of this binary tree are subsets of the input point set P that contain exactly one element each. As a subtree's leaves converge at a certain node, we say that node is internal. We build an instance I of ε-PLSB for any given tree in such a way that performing the query q for I can solve any nearest neighbour query for the point set P. To represent the minimum distance between two sets X and Y, we use the notation dist(X, Y). Here we define hierarchical clustering using this notation.

Definition 4.2 *(Hierarchical clustering).* Given a point set P, its hierarchical clustering is a *binary* tree with singleton subsections of P serving as leaves, with each subsection of P appearing a unique number of times. The junction of the sub-tree leaves at each internal node indicates that node. Thus, every branch in the tree represents a cluster, which is a subset of P. Do not forget that cluster P is the root. Two numbers, rmin(v) and rmax(v), are associated with the following attributes for each internal node v in the tree.

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

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. BallRange$(p, r_1, r_2) = \{b(p, r) | r_1 \leq r \leq r_2\}$. Note that there are infinite number of balls in 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 BallSet$(v) = \varphi$.
- Otherwise, BallSet(v) = BallSet(left(v)) ∪ BallSet(right(v)) ∪ B_l ∪ B_r, where B_l = BallRange(leader(left(v)), $r_{\min}(v)$, $r_{\max}(v)$), B_r = BallRange(leader(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* $b(p, r) \in$ BallSet(v) *is an inner ball of* BallSet(v), *then* $r \neq r_{\min}(u)$, *for* $u \in$ 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* dist$(q, p) = r$.

4.2. Details of the reduction

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

We provide the reduction approach from ε-NNS to ε-PLSB for an lp metric on Rd, given a hierarchical clustering. Three stages might be seen as dividing this procedure. The first thing we do is make an endless supply of balls P1 such that

The answer to the PLSB inquiry q to P1 is a spherical shape, with its centre located around q's ε/7-nearest neighbour. Moving on to the next phase, we create a limited collection of balls P2 from P1 in such a way that the ball centred around the 2ε/3-nearest neighbour of q is the result of the PLSB query q to P2. The outcome of the ε/7-PLSB query q to P2 is a ball that revolves around the ε-nearest neighbour of q in P, as we have shown in our proof.

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 = leader(left(v)) and p_2 = leader(right(v)). By definition, BallSet(v) = (BallSet(left(v)) $\cup$ B_l) $\cup$ (BallSet(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 BallSet(left(v)) $\cup$ $B_l = B_1$ (say) is a set of balls with centers among points of left(v) and BallSet(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 dist$(q, p_1) \geq r_{\max}(v) \geq 8\,\text{diam}(v)/\varepsilon$. So, distance between q and p_1 is large compared to the diameter of v. It is easy to see that for any point $p \in v$, $\frac{\text{dist}(q,p)}{\text{dist}(q,v)} \leq \frac{r_{\max}(v)}{r_{\max}(v) - \text{diam}(v)} \leq \frac{8}{8-\varepsilon} \leq 1 + \varepsilon/7$.

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 dist$(q, p_1^*) \leq r_1$ and dist$(q, p_2^*) \leq r_2$.

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

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

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, $r_1 \le r_2$. So, $b(p_1^*, r_1)$ is the smallest ball containing q, in $B_1 \cup B_2 = \text{BallSet}(v)$.

Also, $\frac{\text{dist}(q,p_1^*)}{\text{dist}(q,\text{left}(v))} \le 1 + \varepsilon/7$ and $\frac{\text{dist}(q,p_2^*)}{\text{dist}(q,\text{right}(v))} \le 1 + \varepsilon/7$ from the Lemma 4.1. Since, p^*_1 is nearer to q than p^*_2 therefore $\frac{\text{dist}(q,p_1^*)}{\text{dist}(q,\text{right}(v))} \le 1 + \varepsilon/7$. Therefore $\frac{\text{dist}(q,p_1^*)}{\min(\text{dist}(q,\text{left}(v)),\text{dist}(q,\text{right}(v)))} \le 1 + \varepsilon/7$. 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. Lemma 4.1 will only be proven for B1 here. The same holds true for B2. We may express it as B1 ∪ BallSet(left(v)) Bl. Assuming that no ball in BallSet(left(v)) includes q, all the points in left(v) are ε/7-nearest neighbours of q according to the induction hypothesis. The smallest ball in BallSet(left(v)) that contains q is centred around an ε/7-nearest neighbour of q in left(v) if q is included in a ball of BallSet(left(v)). Assume that the smallest ball in BallSet(left(v)) that contains q is b(pr,rr). Our task is complete if this is the tiniest ball in B1. Forget about it. From Bl comes the tiniest ball in B1. Take Bl (and by extension, B1) as a whole, and consider the smallest ball b(p1, rrr).

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

Case (ii). $b(p^r, r^r)$ is an inner ball: In this case, r^r is $r_{\min}(v^r)$ for some $v^r \subset v$. The smallest ball in B_l is of radius $r_{\min}(v)$. So $r^{rr} \ge r_{\min}(v) \ge r_{\min}(v^r) = r^r$. So, the $b(p^r, r^r)$ 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 BallRange$(p, r_1, r_2) \subset$ BallSet(P) with

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

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 $(1 + \varepsilon/7)(1 + \varepsilon/7)$ $(\le (1 + 2\varepsilon/5))$ 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 $(1 + \varepsilon/7)^2$ dist(q, P). □

4.2.3. Reduction to ε/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*

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

balls such that following is true for an ε/7-PLSB constructed for it. Given a query point q, the point returned by ε/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 dist$(q, q') \le \varepsilon r/7$ and $b(p, r)$ is the smallest ball in B containing q'. Using this with Theorem 4.2, dist(q, P) + dist$(q, q') \ge$ dist$(q', p)/(1 + \varepsilon/7)^2$. This implies dist$(q, P) \ge$ dist$(q', p)/(1 + \varepsilon/7)^2 - \varepsilon r/7$. Simplifying, we get dist$(q, P) \ge$ 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

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

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

$r_{\min}(v) = \frac{1}{2}$dist left$(v)$, right$(v)$ and
$r_{\max}(v) = 8$ 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 dist$(p_1, p_2) \le$ len(e_1) + len(e_2) + ... + len(e_l). Since, l is at most n and len$(e_i) \le$ dist(left(v), right(v)) (because dist(left(v), right(v)) is the length of the largest edge in MST over v). Therefore, for any $p_1, p_2 \in v$ we have dist$(p_1, p_2) \le n$ dist(left(v), right(v)). So, diameter of v is at most n dist(left(v), right(v)). Thus,

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

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 11, Issue 1, 2023

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

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

Here are some issues with MST hierarchical clustering:

(1) Building the MST is a part of the preprocessing that takes almost quadratic time.

(2) Finding the diameters of point sets is computationally intensive and is part of the preprocessing.

(3) Instead of being linear, the space need is n log n.

Approximate MSTs are used to address the initial issue. We use different interpretations of rmax(v) to deal with the second issue. Before we provide our answer in the following part, let's examine some cases pertaining to the third difficulty.

Ω(P log(P)) is the worst-case space requirement for the current MST hierarchical clustering. A collection of n points p1, p2,..., pn on a straight line with dist(pi, pi 1) = ih is one example. In this case, h is a very little number. A line graph (Fig. 2) represents the MST in this instance, and

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

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

We provide a one-dimensional solution for points in a line case in linear space for the sake of exposition. Ignoring the exact ordering, let's use the prior example based on the fact that all MST edges are almost equal. Building on this concept, we create the completely balanced tree-shaped hierarchical clustering shown in Figure 2 (the second clustering). For this instance,

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

Partitioning the edges of this MST into sets where all the edges inside a set have lengths within a factor of two of each other is the essence of our method. All edges within a single partition are treated as if they were of the same length when building the hierarchical clustering from the approx MST. Our standard MST hierarchical clustering procedure involves sorting the edges in ascending order of length and continuously merging clusters linked by the edges. The sequence of merging will be rearranged here. A lot more work goes into this reordering for the general d-dimensional clustering. A more straightforward approach is shown, nevertheless, for a single dimension.

Let $T = (P, E_M)$ be the MST for P. We partition E_M, into $\{E_k \mid k \in Z\}$, where $E_k = \{e \in E_M \mid 2^k \leq \text{len}(e) < 2^{k+1}\}$. Suppose $E_{k_1}, E_{k_2}, \ldots, E_{k_i}$, are the set of all *non-empty* E_k'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. SupposeWe need to combine the clusters c0, c1, c2,..., ck 1 into one because of the edges in Eki. This is because, as a line graph, the MST naturally forms contiguous intervals, and these clusters would fit exactly into that description. We will assume, without limiting

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

ourselves, that the sequence c0, c1, c2,..., ck1 is the same as it appears on the actual line. Next, we arrange the nodes c0, c1,..., ck 1 in a height-balanced binary tree. This tree suggests a natural sequence for the merging, therefore that's how it happens. Any cluster v that is generated in phase i will have an rmin(v) equal to 2ki −1 for this clustering.

Proof for points in a line case

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

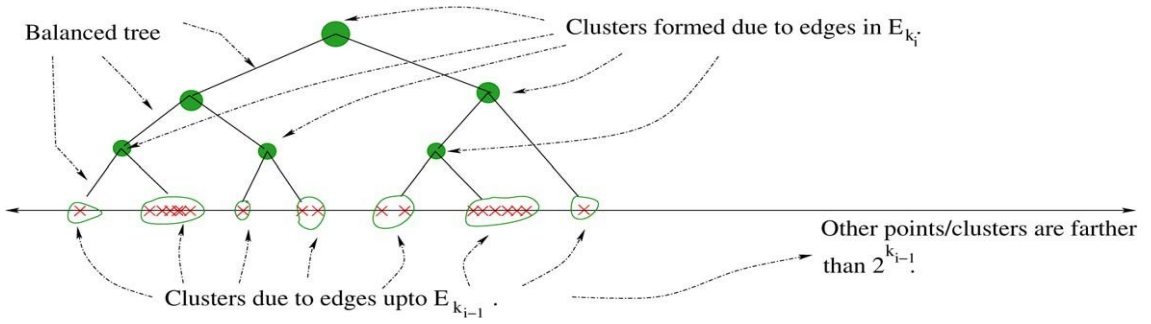

Fig. 4. Linear Hierarchical clustering for points in a line.

(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 $O(\text{len}(e) \sum_{j=i}^{\infty} (\frac{1}{2^{k_j}}))$ $= O(1)$ 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 sethave lengths within a factor of two of each other (actually any constant factor would work). While constructing thehierarchical 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 $\{E_k \mid k \in Z\}$, where $E_k = \{e \in E_M \mid 2^k \leq \text{len}(e) < 2^{k+1}\}$. 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. $F_1 = (\{\{p\} \mid p \in P\}, \varphi)$. 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 withroots 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, $F_1 = (\{\{p\} \mid p \in P\}, \varphi)$. 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) $r_{\min}(v) \leq \frac{1}{2}\text{dist}(\text{left}(v), \text{right}(v))$. *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.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

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 boundedby* 2^{k_i}.

Definition 4.7 *(*$\text{Path}_T(p_1, p_2)$*).* Given a λ-MST T, we define $\text{Path}_T(p_1, p_2)$ 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 *(*$S(v)$*).* Given a λ-MST T, for a $v \subseteq P$, we define

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

Note that if v spans[2] a connected component in T, then $S(v)$ 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 $S(v)$.

In our hierarchical clustering, we shall use

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

Observation 4.4.

$$\text{diam}(v) \le S(v).$$

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 $\alpha = \log_{1+\varepsilon}\left(\frac{\lambda}{\varepsilon}\right)$.

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

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

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 $j \ge i$, it contributes $\alpha\,\text{len}(e)/2^{k_j}$. So, total contribution is at most $\alpha\,\text{len}(e) \sum_{j=i}^{\infty}(1/2^{k_j}) \le 2\alpha\,\text{len}(e)/2^{k_i}$. Also, since $e \in E_{k_i}$, so $\text{len}(e) \le 2^{k_i+1}$. 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 $G' = (S, E \cap (S)_2)$ is a connected graph.

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 11, Issue 1, 2023

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.

After deleting v and reversing the selection of u in T, couple u and v, and then pair the remainder of the vertices using the induction hypothesis. This process may be repeated if u is also to be paired.
To avoid pairing u with v, delete v and choose u using the induction hypothesis. The next step is to pair each vertex with an adjacent vertex in T. Let us pretend that u and w are coupled. After that, keep all of the pairings except for v and w. Every one of the pathways is clearly edge-joint.
We will create an algorithm that pairs k vertices in O(n) time using the aforementioned technique. With a preprocessing time of O(n log(n)), we want an algorithm that can pair the inputs in O(k log(n)) time. You can figure out the features from this sketch of the building. First things first: build an approximate MST balanced edge separator tree. The temporal complexity for this task is O(n log n). Then, we divide the k input points in half along the separator tree sides in order to solve a pairing issue for k points. The next step is to use the corresponding subtree to recursively couple the two components. Lastly, if there are any unpaired vertices in either side, they are joined up with each other to provide a partial solution. The obtained query time 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.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

- 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*

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

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

Proof. From the definition of the star, $c_0 \cup c_j$ spans a connected component in $\bigcup_{j=1}^{i+1} E_{k_j}$. So, there exists an edge in E_k 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.

$$S(c_m \cup c_n) \leq S(c_m) + S(c_n) + \text{PathLength}\ v_m', v_n',$$

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.

$$\text{PathLength}\ v_m', v_n' \leq \text{PathLength}(v_m, v_n) + 2 \times 2^{k_i}.$$

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:

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

Using Observations 4.6 and 4.7, we obtain

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

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.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

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*

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

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 V of T'. Note that the smallest vertex cover of a tree has size at most half the size of the tree $|V| \leq \frac{1}{2}|A|$.
- For each cluster $c_0 \in V$, we have a set of clusters $\{c_1, c_2, \ldots, c_k\} \subset A \setminus V$, 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 V to *partition* A into *stars*. Since, each star is centered around a vertex of the vertex cover, the number of stars is $|V|$.

- Merge the stars using the procedure referred to in Lemma 4.5. After the merging, number of components left is $|V| \leq |A|/2$. 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*

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

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, $k^r = \frac{k}{2^t}$.

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

$$k^r = \frac{k}{2^t}.$$

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,*

$$\sum_{j=1}^{k^r} S(s_j) \le \sum_{j=1}^{k} S(c_j) + t\, S(c_0) + k2^{k_i+1}.$$

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 $O(k^r \log(n))$ time and merging takes $O(k^r)$ time. Since, in the tth iteration, $k^r \le 2k/2^t$, 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

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

Using weighted Geometric mean ≤ weighted arithmetic mean,

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

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

$$\sum_{j=1}^{k^r/2} \log \frac{S(s_{2j-1} \cup s_{2j})}{2^{k_i}} \le \frac{k}{2^{t-1}} \log 2^t \frac{\sum_{j=1}^{k} S(c_j) + tS(c_0) + t2^{k_i+1}}{2^{k_i}k}.$$

So the total contribution is

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

It is easily shown that this is

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

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. □

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 11, Issue 1, 2023

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

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

From (geometric mean ≤ arithmetic mean), we get

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

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

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

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

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

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

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 $S(D)$, $|C| \le \frac{S(D)}{2^{k_i}}$. 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, $|A| \le |C|/2^k$, 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

Two methods were outlined by Har-Peled for estimating MSTs. Building a 1 δ-MST requires an amount of time equal to Od (n log(n) n). Well Separated Pair Decomposition by Callahan Kossaraju is employed. In addition, he provides a method for building a nd-MST that uses O(n log(n)) time. The first method may be used to build a 2-MST that will serve our needs.
Instead of using WSPD, we may build a λ-MST by utilising the methods outlined in this study. This is the way it is done.

(1) Apply the Har-Peled method to construct nd-MST T1 of P. Just so you know, WSPD is not being used here.

(2) Create an estimate of ε μ for Voronoi Diagram V using the methods outlined in this article and T1. It would consume space of order O(n log n).

(3) The Delaunay graph of a Voronoi diagram (or approximation Voronoi diagram) is defined as a graph over the diagram's sites, where edges between points p1 and p2 are defined as cells of p1 and p2 sharing a border, if and only if this formula is true. Define V and build its Delaunay graph F.

G's MST is equal to the 1 + 3μ-MST of the set P of points.

The evidence rests on the fact that an approximation MST is included in an approximate Delaunay graph. When dealing with accurate MST and Delaunay graphs, this may be translated into a comparable outcome. To show this, we will show that for any cut (P1, P2) in P, there is an edge (p1r, p2r) in the Delaunay graph where the distance between the two edges is at most
The distance between points P1 and P2 is (1 + 3μ).

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

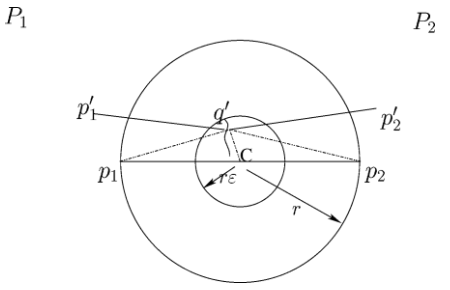

Fig. 5. A cut in an approximate MST.

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^r) 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^r and p_2^r. So, both p_1^r and p_2^r are ε-nearest neighbor's of q^r. So, $\text{dist}(q^r, p_i^r) \leq (1 + \varepsilon)\,\text{dist}(q^r, p_i)$. Using this and from geometry, we get $\text{dist}(p_1^r, p_2^r) \leq \text{dist}(p_1^r, q^r) + \text{dist}(p_2^r, q^r) \leq (1 + \varepsilon)(\text{dist}(p_1, q^r) + \text{dist}(p_2, q^r)) \leq (1 + \varepsilon)(r + r\varepsilon + r + r\varepsilon) \leq (1 + \varepsilon)^2\,\text{dist}(P_1, P_2) \leq (1 + 3\varepsilon)\,\text{dist}(P_1, P_2)$. Since there is an edge in the Delaunay graph between p_1^r and p_2^r, the result follows. □

Appendix C. Hierarchical clustering for construction of PLSBs

Similar to Har-Peled's [11] method, our PLSB creation strategies rely on a hierarchical clustering of the point set. The main difference between both approaches is in the construction of the hierarchical clustering, but otherwise, the concepts are same. To begin, we have the discrete points that make up the hierarchical clustering's leaves. Each stage involves merging two clusters according to a different metric for "proximity" between them. Each of these groups has a delegate who will lead the construction of a set of balls. You may utilise these balls to find the cluster that has a point that is almost closer to your query point. The representation that the ball holding the query point is built around determines this. You don't need to build extra balls to differentiate between the closeness of the query point to points in the two clusters when it's far enough away from both to report a point from either cluster. The procedure is carried out until every point has been combined into one cluster.
The balls in the PLSB issue are the set of all the balls made in this way.

C.1. Answering PLSB using compressed quadtree

D An effective data structure for solving a PLSB issue was outlined by Har-Peled [11] as follows: after estimating the balls using axis-parallel cubes taken from a meticulously built hierarchical grid, a quadtree search tree was built over these cubes. An overview of the data structure is provided below.
Here we have a PLSB issue using m balls. Our goal is to build a data structure that efficiently finds the smallest ball that contains a query point q and reports it. A hierarchical grid of cubes is used to approximate the balls (Fig. 6).
The equation (u) represents the division of d into a uniform axis parallel grid with the origin as its centre and side length equal to 2 log u for a real value u. Changing u's value yields a space-covering multi-resolution grid.

Now think of an r-radius ball with a centre at point p, denoted as b = B(p, r). A collection of cubes of the grid G(r/εd) is denoted as GC(b, ε).

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

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

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

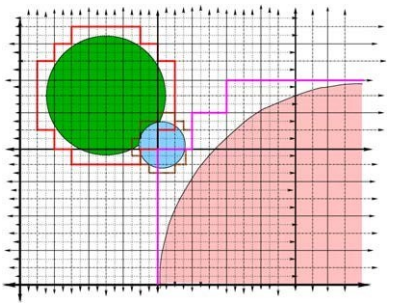

Fig. 6. ε-PLSB data structure.

it encompasses the whole area. When the query point is distant enough from P that any point x P is an ε-closest neighbour to the query point, this backdrop cube is used to answer ε estimated nearest neighbour queries. Keep in mind that the smaller cube, or the cube that corresponds to the smaller ball, always takes precedence whenever two or more cubes from different balls intersect.
This collection contains all the cubes that were created. The cubes may be kept in a quadtree and used to answer point-location queries among the prioritised cubes since they are all obtained from a hierarchical grid representation of d. Areas that make up the compressed quadtree may be either cubes with the same axis or the annulus, or difference, of two cubes that are within each other. The areas encompass space and are not connected to one another. In addition, there is a ball that corresponds to each zone. If there is any overlap, the area is linked to the smallest ball. The conventional algorithms allow for the computation of in O(log n) time. The data structure from [10] may be used to preprocess the leaves of for point-location. The time required to answer a point location query for a point q is thus O(log |Q|) = O(log (m/ε)).

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.