A Sub-linear Time Algorithm for Approximating k-Nearest-Neighbor with Full Quality Guarantee

Hengzhao Ma and Jianzhong Li(✉)

Harbin Institute of Technology, Harbin 150001, Heilongjiang, China
hzma@stu.hit.edu.cn, lijzh@hit.edu.cn

Abstract. In this paper we propose an algorithm for the approximate k-Nearest-Neighbors problem. According to the existing researches, there are two kinds of approximation criteria. One is the distance criterion, and the other is the recall criterion. All former algorithms suffer the problem that there are no theoretical guarantees for the two approximation criteria. The algorithm proposed in this paper unifies the two kinds of approximation criteria, and has full theoretical guarantees. Furthermore, the query time of the algorithm is sub-linear. As far as we know, it is the first algorithm that achieves both sub-linear query time and full theoretical approximation guarantees.

Keywords: Computation geometry · Approximate k-nearest-neighbors

1 Introduction

The k-Nearest-Neighbor (kNN) problem is a well-known problem in theoretical computer science and applications. Let (U, D) be a metric space, then for the input set $P \subseteq U$ of elements and a query element $q \in U$, the kNN problem is to find the k elements with smallest distance to q. Since the exact results are expensive to compute when the size of the input is large [19], and approximate results serve as good as the exact ones in many applications [30], the approximate kNN, kANN for short, draws more research efforts in recent years. There are two kinds of approximation criteria for the kANN problem, namely, the distance criterion and the recall criterion. The distance criterion requires that the ratio between the distance from the approximate results to the query and the distance from the exact results to the query is no more than a given threshold. The recall criterion requires that the size of the intersection of the approximate result set and the exact result set is no less than a given threshold. The formal description will be given in detail in Sect. 2. Next we brief the existing algorithms for the kANN problem to see how these two criteria are considered by former researchers.

This work was supported by the National Natural Science Foundation of China under grant 61732003, 61832003, 61972110 and U1811461.

W. Wu and Z. Zhang (Eds.): COCOA 2020, LNCS 12577, pp. 19–31, 2020.
https://doi.org/10.1007/978-3-030-64843-5_2

The algorithms for the kANN problem can be categorized into four classes. The first class is the tree-based methods. The main idea of this method is to recursively partition the metric space into sub-spaces, and organize them into a tree structure. The K-D tree [6] is the representative idea in this category. It is efficient in low dimensional spaces, but the performance drops rapidly when the number of dimension grows up. Vantage point tree (VP-tree) [31] is another data structure with a better partition strategy and better performance. The FLANN [25] method is a recent work with improved performance in high dimensional spaces, but it is reported that this method would achieve in sub-optimal results [20]. To the best of our knowledge, the tree based methods can satisfy neither the distance nor the recall criterion theoretically.

The second class is the permutation based methods. The idea is to choose a set of pivot points, and represent each data element with a permutation of the pivots sorted by the distance to it. In such a representation, close objects will have similar permutations. Methods using the permutation idea include the MI-File [2] and PP-Index [13]. Unfortunately, the permutation based method can not satisfy either of the distance or the recall criterion theoretically, as far as we know.

The third class is the Locality Sensitive Hashing (LSH) based methods. LSH was first introduced by Indyk et al. [19] for the kANN problem where $k = 1$. Soon after, Datar et al. [11] proposed the first practical LSH function, and since then there came a burst in the theoretical and applicational researches on the LSH framework. For example, Andoni et al. proved the lower bound of the time-space complexities of the LSH based algorithms [3], and devised the optimal LSH function which meets the lower bound [4]. On the other hand, Gao et al. [15] proposed an algorithm that aimed to close the gap between the LSH theory and kANN search applications. See [29] for a survey. The basic LSH based method can satisfy only the distance criterion when $k = 1$ [19]. Some existing algorithms made some progress. The C2LSH algorithm [14] solved the kANN problem with the distance criterion, but it has a constraint that the approximation factor must be a square of an integer. The SRS algorithm [28] is another one aimed at the distance criterion. However, it only has partial guarantee, that is, the results satisfy the distance criterion only when the algorithm terminates on a specific condition.

The forth class is graph based methods. The specific kind of graphs used in this method is the proximity graphs, where the edges in this kind of graph are defined by the geometric relationship of the points. See [23] for a survey. The graph based kANN algorithms usually conduct a navigating process on the proximity graphs. This process selects an vertex in the graph as the start point, and move to the destination point following some specific navigating strategy. For example, Paredes et al. [27] used the kNN graph, Ocsa et al. [26] used the Relative Neighborhood Graph (RNG), and Malkov et al. [22] used the Navigable Small World Graph (NSW) [22]. None of these algorithms have theoretical guarantee on the two approximation criteria.

In summary, most of the existing algorithms do not have theoretical guarantee on either of the two approximation criteria. The recall criterion is only used as a measurement of the experimental results, and the distance criterion is only partially satisfied by only a few algorithms [14,28]. In this paper, we propose a sub-linear time algorithm for kANN problem that unifies the two kinds of approximation criteria, which overcomes the disadvantages of the existing algorithms. The contributions of this paper are listed below.

1. We propose an algorithm that unifies the distance criterion and the recall criterion for the approximate k-Nearest-Neighbor problem. The result returned by the algorithm can satisfy at least one criterion in any situation. This is a major progress compared to the existing algorithms.
2. Assuming the input point set follows the Poisson Point Process, the algorithm takes $O(n \log n)$ time of preprocessing, $O(n \log n)$ space, and answers a query in $O(dn^{1/d} \log n + kn^{\rho} \log n)$ time, where $\rho < 1$ is a constant.
3. The algorithm is the first algorithm for kANN that provides theoretical guarantee on both of the approximation criteria, and it is also the first algorithm that achieves sub-linear query time while providing theoretical guarantees. The former works [14,28] with partial guarantee both need linear query time.

The rest of this paper is organized as follows. Section 2 introduces the definition of the problem and some prerequisite knowledge. The detailed algorithm are presented in Sect. 3. Then the time and space complexities are analyzed in Sect. 4. Finally the conclusion is given in Sect. 5.

2 Preliminaries

2.1 Problem Definitions

The problem studied in this paper is the approximate k-Nearest-Neighbor problem, which is denoted as kANN for short. In this paper the problem is constrained to the Euclidean space. The input is a set P of points where each $p \in P$ is a d-dimensional vector $(p^{(1)}, p^{(2)}, \cdots, p^{(n)})$. The distance between two points p and p' is defined by $D(p, p') = \sqrt{\sum_{i=1}^{d} (p^{(i)} - p'^{(i)})^2}$, which is the well known Euclidean distance. Before giving the definition of the kANN problem, we first introduce the exact kNN problem.

Definition 2.1 (kNN). *Given the input point set $P \subset R^d$ and a query point $q \in R^d$, define $kNN(q, P)$ to be the set of k points in P that are nearest to q. Formally,*

1. *$kNN(q, P) \subseteq P$, and $|kNN(q, P)| = k$;*
2. *$D(p, q) \leq D(p', q)$ for $\forall p \in kNN(q, P)$ and $\forall p' \in P \setminus kNN(q, P)$.*

Next we will give the definition of the approximate kNN. There are two kinds of definitions based on different approximation criteria.

Definition 2.2 ($kANN_c$). *Given the input point set $P \subset R^d$, a query point $q \in R^d$, and a approximation factor $c > 1$, find a point set $kANN_c(q, P)$ which satisfies:*

1. *$kANN_c(q, P) \subseteq P$, and $|kANN_c(q, P)| = k$;*
2. *let $T_k(q, P) = \max_{p \in kNN(q,P)} D(p, q)$, then $D(p', q) \leq c \cdot T_k(q, P)$ holds for $\forall p' \in kANN_c(q, P)$.*

Remark 2.1. The second requirement in Definition 2.2 is called the distance criterion.

Definition 2.3 ($kANN_\delta$). *Given the input point set $P \subset R^d$, a query point $q \in R^d$, and a approximation factor $\delta < 1$, find a point set $kANN_\delta(q, P) \subseteq P$ which satisfies:*

1. *$kANN_\delta(q, P) \subseteq P$, and $|kANN_\delta(q, P)| = k$;*
2. *$|kANN_\delta(q, P) \cap kNN(q, P)| \geq \delta \cdot k$.*

Remark 2.2. If a kANN algorithm returned a set S, the value $\frac{|S \cap kNN(q,P)|}{|kNN(q,P)|}$ is usually called the recall of the set S. This is widely used in many works to evaluate the quality of the kANN algorithm. Thus we call the second statement in Definition 2.3 as the recall criterion.

Next we give the definition of the problem studied in this paper, which unifies the two different criteria.

Definition 2.4. *Given the input point set $P \subset R^d$, a query point $q \in R^d$, and approximation factors $c > 1$ and $\delta < 1$, find a point set $kNN_{c,\delta}(q, P)$ which satisfies:*

1. *$kANN_{c,\delta}(q, P) \subseteq P$, and $|kANN_{c,\delta}(q, P)| = k$;*
2. *$kANN_{c,\delta}(q, P)$ satisfies at least one of the distance criterion and the recall criterion. Formally, either $D(p', q) \leq c \cdot T_k(q, P)$ holds for $\forall p' \in kANN_{c,\delta}(q, P)$, or $|kANN_{c,\delta}(q, P) \cap kNN(q, P)| \geq \delta \cdot k$.*

According to Definition 2.4, the output of the algorithm is required to satisfy one of the two criteria, but not both. It will be our future work to devise an algorithm to satisfy both of the criteria.

In the rest of this section we will introduce some concepts and algorithms that will be used in our proposed algorithm.

2.2 Minimum Enclosing Spheres

The d-dimensional spheres is the generalization of the circles in the 2-dimensional case. Let c be the center and r be the radius. A d-dimensional sphere, denoted as $S(c, r)$, is the set $S(c, r) = \{x \in R^d \mid D(x, c) \leq r\}$. Note that the boundary is included. If $q \in S(c, r)$ we say that q falls inside sphere $S(c, r)$, or the sphere encloses point p. A sphere $S(c, r)$ is said to pass through point p iff $D(c, p) = r$.

Given a set P of points, the minimum enclosing sphere (MES) of P, is the d-dimensional sphere enclosing all points in P and has the smallest possible radius. It is known that the MES of a given finite point set in R^d is unique, and can be calculated by a quadratic programming algorithm [32]. Next we introduce the approximate minimum enclosing spheres.

Definition 2.5 (AMES). *Given a set of points $P \subset R^d$ and an approximation factor $\epsilon < 1$, the approximate minimum enclosing sphere of P, denoted as $AMES(P, \epsilon)$, is a d-dimensional sphere $S(c, r)$ satisfies:*

1. *$p \in S(c, r)$ for $\forall p \in P$;*
2. *$r < (1 + \epsilon)r^*$, where r^* is the radius of the exact MES of P.*

The following algorithm can calculate the AMES in $O(n/\epsilon^2)$ time, which is given in [5].

Algorithm 1: Compute $AMES$

Input: a point set P, and an approximation factor ϵ.
Output: $AMES(P, \epsilon)$

```
c_0 ← an arbitrary point in P;
for i = 1 to 1/ε² do
    p_i ← the point in P farthest away from c_{i−1};
    c_i ← c_{i−1} + (1/i)(p_i − c_{i−1});
end
```

The following Lemma gives the complexity of Algorithm 1 .

Lemma 2.1 [5]**.** *For given ϵ and P where $|P| = n$, Algorithm 1 can calculate $AMES(P, \epsilon)$ in $O(n/\epsilon^2)$ time.*

2.3 Delaunay Triangulation

The Delaunay Triangulation (DT) is a fundamental data structure in computation geometry. The definition is given below.

Definition 2.6 (DT). *Given a set of points $P \subset R^d$, the Delaunay Triangulation is a graph $DT(P) = (V, E)$ which satisfies:*

1. *$V = P$;*
2. *for $\forall p, p' \in P$, $(p, p') \in E$ iff there exists a d-dimensional sphere passing through p and p', and no other $p'' \in P$ is inside it.*

The Delaunay Triangulation is a natural dual of the Voronoi diagram. We omit the details about their relationship since it is not the focus of this paper.

There are extensive research works about the Delaunay triangulation. An important problem is to find the expected properties of DT built on random point sets. Here we focus on the point sets that follow the Poisson Point Process in d-dimensional Euclidean space. In this model, for any region $\mathcal{R} \subset R^d$, the probability that $\mathcal{R}$ contains k points follows a Poisson-like distribution. See [1] for more details. We cite one important property of the Poisson Point Process in the following lemma.

Lemma 2.2 [1]. *Let $S \subset R^d$ be a point set following the Poisson Point Process. Suppose there are two regions $B \subseteq A \subset R^d$. For any point $p \in S$, if p falls inside A then the probability that p falls inside B is the ratio between the volume of B and A. Formally, we have*

$$\Pr[p \in B \mid p \in A] = \frac{volume(B)}{volume(A)}.$$

Further, we cite some important properties of the Delaunay triangulation built on point sets which follow the Poisson Point Process.

Lemma 2.3 [7]. *Let $S \subset R^d$ be a point set following the Poisson Point Process, and $\Delta(G) = \max_{p \in V(G)} |\{(p, q) \in E(G)\}|$ be the maximum degree of G. Then the expected maximum degree of $DT(S)$ is $O(\log n / \log \log n)$.*

Lemma 2.4 [9]. *Let $S \subset R^d$ be a point set following the Poisson Point Process. The expected time to construct $DT(S)$ is $O(n \log n)$.*

2.4 Walking in Delaunay Triangulation

Given a Delaunay Triangulation DT, the points and edges of DT form a set of simplices. Given a query point q, there is a problem to find which simplex of DT that q falls in. There is a class of algorithms to tackle this problem which is called Walking. The Walking algorithm starts at some simplex, and *walks* to the destination by moving to adjacent simplices step by step. There are several kinds of walking strategy, including Jump&Walk [24], Straight Walk [8] and Stochastic Walk [12], etc. Some of these strategies are only applicable to 2 or 3 dimensions, while Straight Walk can generalize to higher dimension. As Fig. 1 shows, the Straight Walk strategy only considers the simplices that intersect the line segment from the start point to the destination. The following lemma gives the complexity of this walking strategy.

Lemma 2.5 [10]. *Given a Delaunay Triangulation DT of a point set $P \subset R^d$, and two points p and p' in R^d as the start point and destination point, the walking from p to p' using Straight Walk takes $O(n^{1/d})$ expected time.*

2.5 (c, r)-NN

The Approximate Near Neighbor problem is introduced in [19] for solving the $kANN_c$ problem with $k = 1$. Usually the Approximate Near Neighbor problem is denoted as (c, r)-NN since there are two input parameters c and r. The definition is given below. The idea to use (c, r)-NN to solve $1ANN_c$ is via Turing reduction, that is, use (c, r)-NN as an oracle or sub-procedure. The details can be found in [16,17,19,21].

Definition 2.7. *Given a point set P, a query point q, and two query parameters $c > 1, r > 0$, the output of the (c, r)-NN problem should satisfy:*

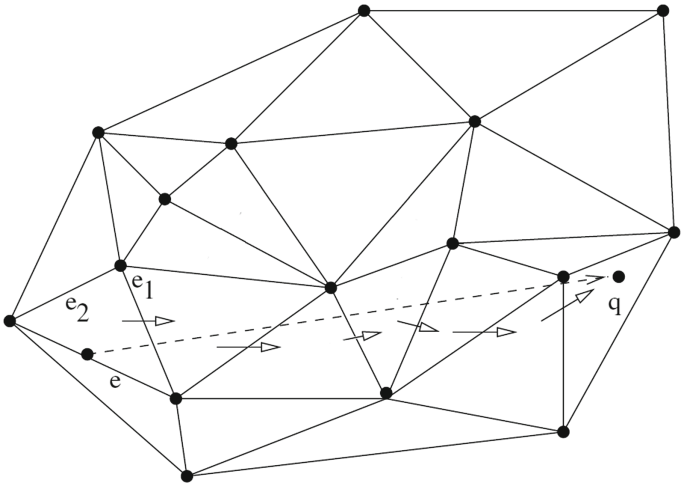

Fig. 1. Illustration of the straight walk

1. *if* $\exists p^* \in S(q,r) \cap P$, *then output a point* $p' \in S(q, c \cdot r) \cap P$;
2. *if* $D(p,q) > c \cdot r$ *for* $\forall p \in P$, *then output No;*

Since we aim to solve kANN problem in this paper, we need the following definition of (c,r)-kNN.

Definition 2.8. *Given a point set* P, *a query point* q, *and two query parameters* c, r, *the output of the* (c,r)*-kNN problem is a set* $kNN_{(c,r)}(q,P)$, *which satisfies:*

1. *if* $|P \cap S(q,r)| \geq k$, *then output a set* $Q \subseteq P \cap S(q, c \cdot r)$, *where* $|Q| = k$;
2. *if* $|P \cap S(q, c \cdot r)| < k$, *then output* $\emptyset$;

It can be easily seen that the (c,r)-kNN problem is a natural generalization of the (c,r)-NN problem. Recently, there are several algorithms proposed to solve this problem. The following Lemma 2.6 gives the complexity of the (c,r)-kNN algorithm. The proof can be found on the online version of this paper [18].

Lemma 2.6. *There is an algorithm that solves* (c,r)*-kNN problem in* $O(kn^{\rho})$ *of time, requiring* $O(kn^{1+\rho} \log n)$ *time of preprocessing and* $O(kn^{1+\rho})$ *of space. The parameter* ρ *is a constant depending on the LSH function used in the algorithm, and* $\rho < 1$ *always holds.*

3 Algorithm

The proposed algorithm consists of two phases, i.e., the preprocessing phase and the query phase. The preprocessing phase is to built a data structure, which will be used to guide the search in the query phase. Next we will describe the algorithm of the two phases in detail.

3.1 Preprocessing Algorithm

Before describing the details of the preprocessing algorithm, we first introduce several concepts that will be used in the following discussion.

Axis Parallel Box. An axis parallel box B in R^d is defined to be the Cartesian product of d intervals, i.e., $B = I_1 \times I_2 \times \cdots \times I_d$. And the following is the definition of Minimum Bounding Box.

Definition 3.1. *Given a point set P, the Minimum Bounding Box, denoted as $MBB(P)$, is the axis parallel box satisfying the following two requirements:*

1. *$MBB(P)$ encloses all points in P, and*
2. *there exists points p and p' in P such that $p^{(i)} = a_i, p'^{(i)} = b_i$ for each interval $I_i = (a_i, b_i)$ defining $MBB(P)$, $1 \leq i \leq d$.*

Median Split. Given a point set P and its minimum bounding box $MBB(P)$, we introduce an operation on P that splits P into two subsets, which is called median split. This operation first finds the longest interval I_i from the intervals defining $MBB(P)$. Then, the operation finds the median of the set $\{p^{(i)} \mid p \in P\}$, which is the median of the i-th coordinates of the points in P. This median is denoted as $med_i(P)$. Finally P is split into two subsets, i.e., $P_1 = \{p \in P \mid p^{(i)} \leq med_i(P)\}$ and $P_2 = \{p \in P \mid p^{(i)} > med_i(P)\}$. Here we assume that no two points share the same coordinate in any dimension. This assumption can be assured by adding some random small shift on the original coordinates.

Median Split Tree. By recursively conducting the median split operation, a point set P can be organized into a tree structure, which is called the Median Split Tree (MST). The definition of MST is given below.

Definition 3.2. *Given the input point set P, a Median Split Tree (MST) based on P, denoted as $MST(P)$, is a tree structure satisfying the following requirements:*

1. *the root of $MST(P)$ is P, and the other nodes in $MST(P)$ are subsets of P;*
2. *there are two child nodes for each interior node $N \in MST(P)$, which are generated by conducting a median split on N;*
3. *each leaf node contains only one point.*

Balanced Median Split Tree. The depth of a node N in a tree T, denoted as $dep_T(N)$, is defined to be the number of edges in the path from N to the root of T. It can be noticed that the leaf nodes in the MST may have different depths. So we introduce the Balanced Median Split Tree (BMST), where all leaf nodes have the same depth.

Let $L_T(i) = \{N \in T \mid dep_T(N) = i\}$, which is the nodes in the i-th layer in tree T, and $|N|$ be the number of points included in node N. For a median split

tree $MST(P)$, it can be easily proved that either $|N| = \lceil n/2^i \rceil$ or $|N| = \lfloor n/2^i \rfloor$ for $\forall N \in L_{MST(P)}(i)$. Given $MST(P)$, the $BMST(P)$ is constructed as follows. Find the smallest i such that $\lfloor n/2^i \rfloor \leq 3$, then for each node $N \in L_{MST(P)}(i)$, connect all the nodes in the sub-tree rooted at N directly to N.

Hierarchical Delaunay Graph. Given a point set P, we introduce the most important concept for the preprocessing algorithm in this paper, which is the Hierarchical Delaunay Graph (HDG). This structure is constructed by adding edges between nodes in the same layer of $BMST(P)$. The additional edges are called the graph edges, in contrast with the tree edges in $BMST(P)$. The definition of the HDG is given below. Here $Cen(N)$ denotes the center of $AMES(N)$.

Definition 3.3. *Given a point set P and the balanced median split tree $BMST(P)$, a Hierarchical Delaunay Graph HDG is a layered graph based on $BMST(P)$, where each layer is a Delaunay triangulation. Formally, for each $N, N' \in HDG(P)$, there is an graph edge between N, N' iff*

1. *$dep_{BMST(P)}(N) = dep_{BMST(P)}(N')$, and*
2. *there exists a d-dimensional sphere S passing through $Cen(N), Cen(N')$, and there is no $N'' \in HDG(P)$ such that $Cen(N'')$ falls in S, where N'' is in the same layer with N and N'. That is, the graph edges connecting nodes in the same layer forms the Delaunay Triangulation.*

The Preprocessing Algorithm. Next we describe the preprocessing algorithm which aims to build the HDG. The algorithm can be divided into three steps.

Step 1, split and build tree. The first step is to recursively split P into smaller sets using the median split operation, and the median split tree is built. Finally the nodes near the leaf layer is adjusted to satisfy the definition of the balanced median split tree.

Step 2, compute spheres. In this step, the algorithm will go over the tree and compute the AMES for each node using Algorithm 1.

Step 3, construct the HDG. In this step, an algorithm given in [9] which satisfies Lemma 2.4 is invoked to compute the Delaunay triangulation for each layer.

The pseudo codes of the preprocessing algorithm is given in Algorithm 2.

3.2 Query Algorithm

The query algorithm takes the HDG built by the preprocessing algorithm, and executes the following three steps.

The first is the descending step. The algorithm goes down the tree and stops at level i such that $k \leq n/2^i < 2k$. At each level, the child node with smallest distance to the query is chosen to be visited in next level.

Algorithm 2: Preprocessing Algorithm

Input: a point set P
Output: a hierarchical Delaynay graph $HDG(P)$
$T \leftarrow$`SplitTree(`P`)`;
Modify T into a BMST;
`ComputeSpheres(`T`)`;
`HierarchicalDelaunay(`T`)`;
Procedure `SplitTree(`N`)`**:**
  Conduct median split on N and generate two sets N_1 and N_2;
  $T_1 \leftarrow$`SplitTree(`N_1`)`;
  $T_2 \leftarrow$`SplitTree(`N_2`)`;
  Let T_1 be the left sub-tree of N, and T_2 be the right sub-tree of N;
end
Procedure `ComputeSpheres(`T`)`**:**
  foreach $N \in T$ **do**
    Call `AMES(`$N, 0.1$`)` (Algorithm 1);
  end
end
s **Procedure** `HierarchicalDelaunay(`T`)`**:**
  Let dl be the depth of the leaf node in T;
  for $i = 0$ *to* dl **do**
    `Delaunay(`$L_T(i)$`)` (Lemma 2.4);
  end
end

The second is the navigating step. The algorithm marches towards the local nearest AMES center by moving on the edges of the HDG.

The third step is the answering step. The algorithm finds the answer of $kANN_{c,\delta}(q, P)$ by invoking the (c, r)-kNN query. The answer can satisfy the distance criterion or the recall criterion according to the different return result of the (c, r)-kNN query.

Algorithm 3 describes the above process in pseudo codes, where $Cen(N)$ and $Rad(N)$ are the center and radius of the $AMES$ of node N, respectively.

4 Analysis

The analysis in this section will assume that the input point set P follows the Poisson Point Process. The proofs can be found in the online version of this paper [18], and are all omitted here due to space limitation.

4.1 Correctness

Lemma 4.1. *If Algorithm 3 terminates when $i = 0$, then the returned point set Res is a δ-kNN of q in P with at least $1 - e^{-\frac{n-k}{n^d}}$ probability.*

Algorithm 3: Query

Input: a query points q, a point set P, approximation factors $c > 1, \delta < 1$, and $HDG(P)$
Output: $kANN_{c,\delta}(q, P)$

$N \leftarrow$ the root of $HDG(P)$;
while $|N| > 2k$ **do**
  $Lc \leftarrow$ the left child of N, $Rc \leftarrow$ the right child of N;
  if $D(q, Cen(Lc)) < D(q, Cen(Rc))$ **then**
    $N \leftarrow Lc$;
  else
    $N \leftarrow Rc$;
  end
end
while $\exists N' \in Nbr(N)$ *s.t.* $D(q, Cen(N')) < D(q, Cen(N))$ **do**
  $N \leftarrow \arg \min_{N' \in Nbr(N)} \{D(q, Cen(N'))\}$;
end
for $i = 0$ *to* $\log_c n$ **do**
  Invoke (c, r)-kNN query where $r = \frac{D(q,Cen(N))+Rad(N)}{n} c^i$;
  if *the query returned a set Res* **then**
    return *Res* as the final result;
  end
end

Lemma 4.2. *If Algorithm 3 terminates when $i > 0$, then the returned point set Res is a c-kNN of q in P.*

Theorem 4.1. *The result of Algorithm 3 satisfies the requirement of $kNN_{c,\delta}(q, P)$ with at least $1 - e^{-\frac{n-k}{n^d}}$ probability.*

4.2 Complexities

Theorem 4.2. *The expected time complexity of Algorithm 2, which is the preprocessing time complexity, is $O(n \log n)$.*

Theorem 4.3. *The space complexity of Algorithm 2 is $O(n \log n)$.*

Theorem 4.4. *The time complexity of Algorithm 3, which is the query complexity, is $O(dn^{1/d} \log n + kn^{\rho} \log n)$, where $\rho < 1$ is a constant.*

5 Conclusion

In this paper we proposed an algorithm for the approximate k-Nearest-Neighbors problem. We observed that there are two kinds of approximation criteria in the history of this research area, which is called the distance criterion and the recall criterion in this paper. But we also observed that all existing works do not

have theoretical guarantees on the two criteria. We raised a new definition for the approximate k-Nearest-Neighbor problem which unifies the distance criterion and the recall criterion, and proposed an algorithm that solves the new problem. The result of the algorithm can satisfy at least one of the two criteria. In our future work, we will try to devise new algorithms that can satisfy both of the criteria.

References

1. Poisson Point Process. https://wikimili.com/en/Poisson_point_process
2. Amato, G., Gennaro, C., Savino, P.: MI-file: using inverted files for scalable approximate similarity search. Multimed. Tools Appl. **71**(3), 1333–1362 (2012). https://doi.org/10.1007/s11042-012-1271-1
3. Andoni, A., Laarhoven, T., Razenshteyn, I., Waingarten, E.: Optimal hashing-based time-space trade-offs for approximate near neighbors. In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 47–66, January 2017
4. Andoni, A., Razenshteyn, I.: Optimal data-dependent hashing for approximate near neighbors. In: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing - STOC 2015, pp. 793–801 (2015)
5. Bâdoiu, M., Bâdoiu, M., Clarkson, K.L., Clarkson, K.L.: Smaller core-sets for balls. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 801–802 (2003)
6. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (1975)
7. Bern, M., Eppstein, D., Yao, F.: The expected extremes in a delaunay triangulation. Int. J. Comput. Geom. Appl. **01**(01), 79–91 (1991)
8. Bose, P., Devroye, L.: On the stabbing number of a random Delaunay triangulation. Comput. Geom.: Theory Appl. **36**(2), 89–105 (2007)
9. Buchin, K., Mulzer, W.: Delaunay triangulations in O(sort(n)) time and more. In: Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS , vol. 5, pp. 139–148 (2009)
10. de Castro, P.M.M., Devillers, O.: Simple and efficient distribution-sensitive point location in triangulations. In: 2011 Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 127–138, January 2011
11. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Twentieth Annual Symposium on Computational Geometry - SCG 2004, pp. 253–262 (2004)
12. Devillers, O., Pion, S., Teillaud, M.: Walking in a triangulation. Int. J. Found. Comput. Sci. **13**(02), 181–199 (2002)
13. Esuli, A.: Use of permutation prefixes for efficient and scalable approximate similarity search. Inf. Process. Manage. **48**(5), 889–902 (2012)
14. Gan, J., Feng, J., Fang, Q., Ng, W.: Locality-sensitive hashing scheme based on dynamic collision counting. In: Proceedings of the 2012 International Conference on Management of Data - SIGMOD 2012, pp. 541–552 (2012)
15. Gao, J., Jagadish, H., Ooi, B.C., Wang, S.: Selective hashing: closing the gap between radius search and k-NN Search. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 2015, pp. 349–358 (2015)

16. Har-Peled, S.: A replacement for Voronoi diagrams of near linear size. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, pp. 94–103. IEEE (2001)
17. Har-Peled, S., Indyk, P., Motwani, R.: Approximate nearest neighbor: towards removing the curse of dimensionality. Theory Comput. **8**(1), 321–350 (2012)
18. Hengzhao Ma, J.L.: A sub-linear time algorithm for approximating k-nearest-neighbor with full quality guarantee (2020). https://arxiv.org/abs/2008.02924
19. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing - STOC 1998, pp. 604–613 (1998)
20. Lin, P.C., Zhao, W.L.: Graph based Nearest Neighbor Search: Promises and Failures, pp. 1–8 (2019)
21. Ma, H., Li, J.: An algorithm for reducing approximate nearest neighbor to approximate near neighbor with O(log n) query time. In: 12th International Conference on Combinatorial Optimization and Applications - COCOA 2018, pp. 465–479 (2018)
22. Malkov, Y., Ponomarenko, A., Logvinov, A., Krylov, V.: Approximate nearest neighbor algorithm based on navigable small world graphs. Inf. Syst. **45**, 61–68 (2014)
23. Mitchell, J.S., Mulzer, W.: Proximity algorithms. In: Handbook of Discrete and Computational Geometry, Third Edition, pp. 849–874 (2017)
24. Mücke, E.P., Saias, I., Zhu, B.: Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. Comput. Geom. **12**(1–2), 63–83 (1999)
25. Muja, M., Lowe, D.G.: Scalable nearest neighbor algorithms for high dimensional data. IEEE Trans. Pattern Anal. Mach. Intell. **36**(11), 2227–2240 (2014)
26. Ocsa, A., Bedregal, C., Cuadros-vargas, E., Society, P.C.: A new approach for similarity queries using proximity graphs, pp. 131–142. Simpósio Brasileiro de Banco de Dados (2007)
27. Paredes, R., Chávez, E.: Using the k-nearest neighbor graph for proximity searching in metric spaces. In: International Symposium on String Processing and Information Retrieval, pp. 127–138 (2005)
28. Sun, Y., Wang, W., Qin, J., Zhang, Y., Lin, X.: SRS: solving c-approximate nearest neighbor queries in high dimensional Euclidean space with a tiny index. Proc. VLDB Endow. **8**, 1–12 (2014)
29. Wang, J., Shen, H.T., Song, J., Ji, J.: Hashing for similarity search: a survey. In: ArXiv:1408.2927 (2014)
30. Weber, R., Schek, H.J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proceedings of 24rd International Conference on Very Large Data Bases, pp. 194–205 (1998)
31. Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 311–321 (1993)
32. Yildirim, E.A.: Two algorithms for the minimum enclosing ball problem. SIAM J. Optim. **19**(3), 1368–1391 (2008)