

An Algorithm for Reducing Approximate Nearest Neighbor to Approximate Near Neighbor with $O(\log n)$ Query Time

Hengzhao Ma[†] and Jianzhong Li[‡]

Harbin Institute of Technology, Harbin, Heilongjiang 150001, China

[†] hzma@stu.hit.edu.cn

[‡] lijzh@hit.edu.cn

Abstract. This paper proposes a new algorithm for reducing Approximate Nearest Neighbor problem to Approximate Near Neighbor problem. The advantage of this algorithm is that it achieves $O(\log n)$ query time. As a reduction problem, the query time complexity is the times of invoking the algorithm for Approximate Near Neighbor problem. All former algorithms for the same reduction need $\text{polylog}(n)$ query time. A box split method proposed by Vaidya is used in our paper to achieve the $O(\log n)$ query time complexity.

Keywords: Computation Geometry · Approximate Nearest Neighbor · Reduction

1 Introduction

The approximate nearest neighbor problem, ϵ -NN for short, can be defined as follows: given a set P of points in a metric space S equipped with a distance function D , and a query point $q \in S$, find a point $p \in P$ such that $D(p, q) \leq (1+\epsilon)D(p^*, q)$, where p^* has the minimal distance to q in P . ϵ -NN is one of the most important proximity problems in computation geometry. Many proximity problems in computation geometry can be reduced to ϵ -NN [12], such as approximate diameter, approximate furthest neighbor, and so on. ϵ -NN is also important in many other areas, such as databases, data mining, information retrieval and machine learning.

Due to its importance, ϵ -NN has been the subject of substantial research efforts. Many algorithms for solving ϵ -NN have been discovered. These works can be summarized into four classes.

The first class of the algorithms tries to build data structures that support solving ϵ -NN efficiently. Arya et. [5] give a such algorithm with query time $1/\epsilon^{O(d)} \cdot \log n$, space $1/\epsilon^{O(d)} \cdot n$ and preprocessing time $1/\epsilon^{O(d)} \cdot n \log n$. Another work [6] gives an algorithm requiring $O(dn)$ space and $O(dn \log n)$ preprocessing time but query time as high as $(d/\epsilon)^{O(d)} \cdot \log n$. Kleinberg proposes two algorithms in [16]. The first algorithm is deterministic and achieves query time of $O(d \log^2 d(d + \log n))$, using a data structure that requires $O((n \log d)^{2d})$ space

and $O((n \log d)^{2d})$ preprocessing time. The second algorithm is a randomized version of the first one. By a preprocessing procedure that takes $O(d^2 \log^2 d \cdot n \log^2 n)$ time, it reduces the storage requirement to $O(dn \cdot \log^3 n)$, but raises the query time up to $O(n + d \log^3 n)$.

The second class of the algorithms considers the situation of $\epsilon = d^{O(1)}$. One such algorithm is given in [7]. It can answer $O(\sqrt{d})$ -NN in $O(2^d \log n)$ time with $O(d8^d n \log n)$ preprocessing time and $O(d2^d n)$ space. Chan [9] improves this result by giving an algorithm that can answer $O(d^{3/2})$ -NN in $O(d^2 \log n)$ query time with $O(d^2 n \log n)$ preprocessing time and $O(dn \log n)$ space.

The third interesting class of work tries to solve ϵ -NN by inspecting some intrinsic dimension of the input point set P . An exemplar work is in [17]. The paper gives an algorithm whose query time is bounded by $2^{O(\dim(P))} \log \Delta + (1/\epsilon)^{O(\dim(P))}$, where $\dim(P)$ is the intrinsic dimension of the input point set P , and Δ is the diameter of P .

Besides these algorithms mentioned above, Indyk et. [15] initiate the work on the fourth class of algorithms. The key idea is to define an Approximate *Near Neighbor* problem, denoted as (c, r) -NN, and reduce ϵ -NN to it. The (c, r) -NN problem can be viewed as a decisive version of ϵ -NN. The formal definition of (c, r) -NN is given in Definition 2 in the next section.

To use this method to solve ϵ -NN, two parts of problem must be considered. One is how to solve (c, r) -NN, and the other is how to reduce ϵ -NN to (c, r) -NN. Some works about the two parts of problem are discussed below. Our study focuses on the latter part.

Algorithms to solve (c, r) -NN The existing algorithms for (c, r) -NN mainly consider the specific situation of d -dimensional Euclidean space with 1-order and 2-order Minkowski distance metrics. Each input point x is given in the form of (x_1, \dots, x_d) . And q -order Minkowski L_q distance between points x and y is given

by $D(x, y) = \left(\sum_{i=1}^d |x_i - y_i|^q \right)^{\frac{1}{q}}$. The 1-order and 2-order Minkowski distance are well-known Manhattan distance and Euclidean distance, respectively. Another simpler situation, which is the (c, r) -NN problem under Hamming cube $\{0, 1\}^d$ equipped with Hamming distance, is usually considered in theoretical studies.

Table 1 summarizes the complexities of the existing algorithms for (c, r) -NN under Euclidean space and L_1 distance. These papers also give solutions under L_2 distance, but we omit these results due to space limitation. Usually the complexities under L_2 distance is higher than that under L_1 distance. It is a key characteristic of the existing algorithms for (c, r) -NN that they usually have different complexities for problems under different order of Minkowski distance metrics.

The listed solutions in Table 1 can be divided into three groups. The first group includes the one given in [15], which is deterministic, and the other groups are randomized. The advantage of randomization is that the exponential complexity about d is freed. The second group includes the one given in [18], which is based on a random projection method proposed in [16]. One distinguished

Table 1. Solutions to (c, r) -NN under Euclidean space and L_1 distance.

Source	Data structure building		Query		Space	Update time
	Time	Failure probability	Time	Failure probability		
[15] ($\epsilon = c - 1$)	$O(n \cdot \frac{1}{\epsilon^d})$	0	$O(1)$	0	$O(n \cdot \frac{1}{\epsilon^d})$	$O(\frac{1}{\epsilon^d})$
[18] ($\epsilon = c - 1$)	$O(n^{\frac{d}{c-1}}(n \log d)^{O(\frac{1}{c-1})})$	$O(1)$	$O(\frac{d}{\epsilon^2} \text{polylog}(dn) \cdot \log \frac{1}{f})$	f	$O(\frac{d^2}{\epsilon^2}(n \log d)^{O(\frac{1}{c-1})})$	$O(n^{O(\frac{1}{c-1})})$
[19]	$O(n^{(\frac{1}{c-1})^2} \log n)$	0	$O(dn^{o(1)})$	$O(1)$	$O(n^{(\frac{1}{c-1})^2})$	$O(n^{(\frac{1}{c-1})^2})$
[10,3,4]	$O(dn^{1+\frac{1}{2c-1}} \log n)$	0	$O(dn^{\frac{1}{2c-1}})$	$O(1)$	$O(dn + n^{1+\frac{1}{2c-1}})$	$O(dn^{\frac{1}{2c-1}+o(1)})$
[1]	$O(dn^{1+o(1)} \log n)$	0	$O(n^{\frac{1}{c-2}})$	$O(1)$	$O(dn^{1+o(1)})$	$O(dn^{o(1)})$

characteristic of the method is that the data structure building stage is also randomized. The last group includes a long line of research work based on Locality Sensitive Hashing (LSH), which is first proposed in [15]. These works are summarized into three terms in Table 1, which can be viewed as the space-time trade-off under LSH framework.

Finally, comparing the five results in Table 1, it can be seen that the query time grows and the space requirement drops from the first one to the last. The five results form a general space-time trade-off about the solution to (c, r) -NN.

Reducing ϵ -NN to (c, r) -NN So far there are three different algorithms for such a reduction. Two of the three algorithms are deterministic [15,13], and the other one is randomized [14]. The complexities of the three reduction algorithms are summarized in Table 2. Note that query time in Table 2 is the number of invocations of (c, r) -NN algorithm. And the preprocessing time about [15] is not given because there is no such analysis in that paper.

Table 2. Comparison of three reductions.

Source	Approximation factor	Preprocessing		Query		Space
		Time	Failure probability	Time (# of (c, r) -NN invoked)	Failure probability	
[14]	$\frac{c(1+\gamma)^2}{\gamma \log^2 n}$ ($\gamma \in (\frac{1}{n}, \frac{1}{2})$) ($c = 1 + \epsilon$)	$O(\frac{T(n, c, f)}{\gamma \log^2 n} + n \log n [Q(n, c, f) + D(n, c, f)])$	$f \log n$	$O(\log^{O(1)} n)$	$f \log n$	$O(\frac{S(n, c, f)}{\gamma \log^2 n})$
[15]	$1 + \epsilon$	-	-	$O(\log^2 n)$	0	$O(n \cdot \text{polylog}(n))$
[13]	$1 + \epsilon$	$O(d \cdot n^{\frac{\log n}{c-1}} \log \frac{n}{\epsilon})$	0	$O(\log \frac{n}{\epsilon})$	0	$O(d \cdot n^{\frac{\log n}{c-1}} \log \frac{n}{\epsilon})$

Among the three reduction algorithms, the one proposed in [14] need to be explained in detail. First, the algorithm outputs a point p' such that $D(q, p') \leq c(1 + \gamma)^2 D(q, p^*)$, where $c = 1 + \epsilon$ and p^* is the exact NN of q . Second, the $T(n, c, f)$, $Q(n, c, f)$, $D(n, c, f)$ and $S(n, c, f)$ functions represent the complexity functions of the data structure building time, query time, update time and storage usage for (c, r) -NN, respectively. Third, the parameter f is the failure probability of one (c, r) -NN invocation, and is selected so that $f \log n$ is a constant less than 1.

The fourth and the most important point about [14] is the $O(\log^{O(1)} n)$ query time. The algorithm given in [14] explicitly invokes $O(\log n)$ times of (c, r) -NN, and each invocation needs $T(n, c, f)$ time. As explained above, the parameter f , which is the failure probability of one (c, r) -NN invocation, is set to $O(\frac{1}{\log n})$. Note that the algorithms for (c, r) -NN given in Table 1 all have constant failure probability¹. In order to satisfy the requirement of $O(\frac{1}{\log n})$ failure probability of one (c, r) -NN invocation, each time the algorithm in [14] invokes (c, r) -NN, the algorithms for (c, r) -NN with constant failure probability must be executed multiple times, which is $O(\log^{O(1)} n)$ times in expectation. Multiplying $O(\log n)$ invocations of (c, r) -NN and $O(\log^{O(1)} n)$ executions of (c, r) -NN algorithm for each invocation, we obtain that the algorithm in [14] actually invokes $O(\log^{O(1)} n)$ times of (c, r) -NN algorithm. This observation is confirmed in [2].

Our method We propose a new algorithm in this paper for reducing ϵ -NN to (c, r) -NN. Comparing with the former works [14, 15, 13], our algorithm has the following characteristics:

(1) It achieves $O(\log n)$ query time, counted in the number of invocations of (c, r) -NN algorithm. It is superior to all the other three works. This is the most distinguished contribution of this paper.

(2) Its preprocessing time is $O((\frac{d}{\epsilon})^d \cdot n \log n)$, and the space complexity is $O((\frac{d}{\epsilon})^d \cdot n)$. Our method has better complexity than the other three works in terms of n , so that it is much suitable to big data with low or fixed dimensionality. This situation is plausible in many applications like road-networks and so on.

(3) In terms of the parameterized complexity treating d as a constant, our result is the closest to the well recognized *optimal* complexity claimed in [6], which requires $O(n \log n)$ preprocessing time, $O(n)$ space and $O(\log n)$ query time.

Note that there is an $O((d/\epsilon)^d)$ factor in our preprocessing and space complexity. This factor originates from a lemma we used in [20]. We point out that the upper bound $O((d/\epsilon)^d)$ is actually very loose. There really is possibility to reduce the upper bound, and thus make our result more close to optimal. In this sense, our work is more promising than all the other three works. However, reducing the upper bound $O((d/\epsilon)^d)$ is out of this paper's scope, and is left as our future work.

2 Problem Definitions and Mathematical Preparations

2.1 Problem definitions

We focus on ϵ -NN in euclidean space R^d . The input is a set P of n points extracted from R^d and a distance metric L_q . Each point x is given as the form

¹ The deterministic one has exponential dependence on d , so it is rarely used in theory and practice.

(x_1, \dots, x_d) . L_q distance metric between points x and y is given by $D(x, y) = \left(\sum_{i=1}^d |x_i - y_i|^q \right)^{\frac{1}{q}}$.

Denote $B(p, r)$ to be the d -dimensional ball centered at p and with radius r . And let $p' \in B(p, r)$ be equivalent to $D(p', p) \leq r$. We first give the definitions of ϵ -NN and (c, r) -NN problems.

Definition 1 (ϵ -NN). *Given a set P of points extracted from R^d , a query point $q \in R^d$, and an approximation factor ϵ , find a point $p' \subseteq P$ such that $D(p', q) \leq (1 + \epsilon)D(p^*, q)$ where $D(p^*, q) = \min_{p \in P} \{D(p, q)\}$.*

Remark 1. p^* is called the nearest neighbor (NN), or exact NN to q , and p' is called an ϵ -NN to q .

Definition 2 ((c, r) -NN). *Given a set P of points extracted from R^d , a query point $q \in R^d$, a query range r , and an approximation factor $c > 1$, (c, r) -NN problem is to design an algorithm satisfying these:*

1. *if there is a point $p_0 \in P$ satisfying $p_0 \in B(q, r)$, then return a point $p' \in P$ such that $p' \in B(q, c \cdot r)$;*
2. *if $D(p, q) > c \cdot r$ for $\forall p \in P$, then return No.*

Remark 2. There are multiple names referring to the same problem defined above. In the papers related to LSH, it is referred as (c, r) -NN. In [15], it is called approximate Point Location in Equal Balls, which is denoted as ϵ -PLEB where $\epsilon = c - 1$. In more recent papers like [13], it is called Approximate Near Neighbor problem.

Next we give the definition of the reduction problem to be solved in this paper, i.e., the problem of reducing ϵ -NN to (c, r) -NN.

Definition 3 (Reduction Problem). *Given a set P of points extracted from R^d , a query point $q \in R^d$, an approximation factor ϵ , and an algorithm \mathcal{A} for (c, r) -NN, the reduction problem is to find an ϵ -NN to q by invoking the algorithm \mathcal{A} as an oracle.*

Remark 3. To solve the reduction problem, a preprocessing phase is usually needed, which is to devise a data structure \mathcal{D} based on P . Thus the problem of reducing ϵ -NN to (c, r) -NN is divided into two phases. The first is data structure building phase, or preprocessing phase. The second is ϵ -NN searching phase, or query phase. The (c, r) -NN algorithm \mathcal{A} is invoked in query phase as an oracle, which characterizes the algorithm as a Turing reduction from ϵ -NN to (c, r) -NN.

The time complexity of the algorithm for the reduction problem consists of two parts, namely, preprocessing time complexity and query time complexity. An important note is that the query time complexity is the number of invocations of (c, r) -NN algorithm \mathcal{A} . This is the well recognized method for analyze the time complexity of a Turing reduction.

2.2 Mathematical Preparations

In this section we introduce some denotations and lemmas to support the idea of our algorithm for reducing ϵ -NN to (c, r) -NN.

Denotations Define a box \mathbf{b} in R^d to be the product of d intervals, i.e., $I_1 \times I_2 \times \dots \times I_d$ where I_i is either open, closed or semi-closed interval, $1 \leq i \leq m$. A box is cubical iff all the d intervals defining the box are of the same length. The side length a cubical box, which is the length of any interval defining the cubical box, is denoted as $len(\mathbf{b})$. A minimal cubical box (MCB) for a point set P , denoted as $MCB(P)$, is the cubical box containing all the points in P and has the minimal side length. Note that $MCB(P)$ may not be unique.

Given a point set P and a box \mathbf{b} , let $p \in \mathbf{b}$ denote that a point $p \in P$ falls inside box \mathbf{b} , and let $|\mathbf{b} \cap P|$ denote the number of points in P that falls inside \mathbf{b} . We will use $|\mathbf{b}|$ for short, if not causing ambiguity.

Given a collection of MCBs $\mathcal{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_m\}$, define $D_{max}(\mathbf{b})$, $D_{min}(\mathbf{b}, \mathbf{b}')$, $D_{max}(\mathbf{b}, \mathbf{b}')$ as follows:

$$D_{max}(\mathbf{b}) = \max_{p_1, p_2 \in \mathbf{b}} D(p_1, p_2), \forall \mathbf{b} \in \mathcal{B}$$

$$D_{min}(\mathbf{b} = \mathbf{b}') = \min_{p \in \mathbf{b}, p' \in \mathbf{b}'} \{D(p, p')\}, D_{max}(\mathbf{b}, \mathbf{b}') = \max_{p \in \mathbf{b}, p' \in \mathbf{b}'} \{D(p, p')\}, \forall \mathbf{b}, \mathbf{b}' \in \mathcal{B}$$

With the above denotations, define $Est(\mathbf{b})$ as follows:

$$Est(\mathbf{b}) = \begin{cases} D_{max}(\mathbf{b}), & \text{if } |\mathbf{b} \cap P| \geq 2 \\ \min_{\mathbf{b}' \in Nbr(\mathbf{b})} \{D_{max}(\mathbf{b}, \mathbf{b}')\}, & \text{otherwise} \end{cases} \quad (1)$$

where $Nbr(\mathbf{b}) = \{\mathbf{b}' \mid D_{min}(\mathbf{b}, \mathbf{b}') \leq r\}$, and the parameter r should satisfy $r \geq Est(\mathbf{b})$.²

For an MCB \mathbf{b} , we associate a ball with it. Pick an arbitrary point $c_{\mathbf{b}} \in \mathbf{b}$, and let $r_{\mathbf{b}} = Est(\mathbf{b})$, then we have a ball $B(c_{\mathbf{b}}, r_{\mathbf{b}})$. It is easily verified that every point in \mathbf{b} is within a distance of $Est(\mathbf{b})$ from $c_{\mathbf{b}}$, in another way to say, the ball $B(c_{\mathbf{b}}, r_{\mathbf{b}})$ encloses every point in \mathbf{b} . We call $B(c_{\mathbf{b}}, r_{\mathbf{b}})$ the *enclosing ball* for box \mathbf{b} .

Next we start to introduce the lemmas while discussing different situations of ϵ -NN search. In the following discussion, we will assume that we have an MCB \mathbf{b} of the input point set P , an enclosing ball $B(c_{\mathbf{b}}, r_{\mathbf{b}})$ of the MCB \mathbf{b} , and a query point q .

Situation 1 The first and an easy situation is that, if q is far enough from $c_{\mathbf{b}}$ then every point in \mathbf{b} is an ϵ -NN to q . The following value $T_1(\mathbf{b})$ explains the threshold for *far enough*, and Lemma 1 depicts the situation discussed above.

² It can be verified that, as long as $r \geq Est(\mathbf{b})$ is satisfied, the value of r doesn't influence the value of $Est(\mathbf{b})$. The specific value of r will be shown latter.

Definition 4. For an MCB of a point set P , define $T_1(\mathbf{b}) = (1 + 2/\epsilon)r_{\mathbf{b}}$.

Lemma 1. If $D(q, c_{\mathbf{b}}) \geq T_1(\mathbf{b})$, then every point in \mathbf{b} is an ϵ -NN to q .

Proof. If $|\mathbf{b}| = 1$ then the lemma is trivial. Assume $|\mathbf{b}| \geq 2$. Starting from the given condition, we first prove $(1 + \epsilon)(D(q, c_{\mathbf{b}}) - r_{\mathbf{b}}) \geq D(q, c_{\mathbf{b}}) + r_{\mathbf{b}}$ as follows:

$$\begin{aligned} D(q, c_{\mathbf{b}}) &\geq (1 + 2/\epsilon)r_{\mathbf{b}} \Rightarrow \\ \epsilon \cdot D(q, c_{\mathbf{b}}) &\geq (\epsilon + 2)r_{\mathbf{b}} \Rightarrow \\ (1 + \epsilon)D(q, c_{\mathbf{b}}) &\geq D(q, c_{\mathbf{b}}) + (\epsilon + 2)r_{\mathbf{b}} \Rightarrow \\ (1 + \epsilon)(D(q, c_{\mathbf{b}}) - r_{\mathbf{b}}) &\geq D(q, c_{\mathbf{b}}) + r_{\mathbf{b}}. \end{aligned}$$

Let the minimal distance from the query point q to a point in \mathbf{b} be $D(q, \mathbf{b})$. Clearly, we have $D(q, \mathbf{b}) \geq D(q, c_{\mathbf{b}}) - r_{\mathbf{b}}$. Then we have $D(q, p) \leq D(q, c_{\mathbf{b}}) + r_{\mathbf{b}} \leq (1 + \epsilon)(D(q, c_{\mathbf{b}}) - r_{\mathbf{b}}) \leq (1 + \epsilon)D(q, \mathbf{b})$ for $\forall p \in \mathbf{b}$. This indicates that every point $p \in \mathbf{b}$ is ϵ -NN to q . \square

Situation 2 If q is not as far from $c_{\mathbf{b}}$ as a distance of $T_1(\mathbf{b})$, i.e., $D(q, c_{\mathbf{b}}) < T_1(\mathbf{b})$, then we split \mathbf{b} into a set of sub-boxes $\{\mathbf{b}_1, \dots, \mathbf{b}_m\}$, and calculate the enclosing balls $B(c_{\mathbf{b}_i}, r_{\mathbf{b}_i})$ for each box $\mathbf{b}_i, 1 \leq i \leq m$. The next situation is that if q is still far enough from each point in $\{c_{\mathbf{b}_1}, \dots, c_{\mathbf{b}_m}\}$, i.e., the centers of the enclosing balls, then we can still tell that every point in \mathbf{b} is an ϵ -NN to q . We give another threshold $T_2(\mathbf{b})$ based on this idea, and formalize the idea into Lemma 2. This lemma also discusses the quantitative relationship between $T_2(\mathbf{b})$ and $T_1(\mathbf{b})$.

Definition 5. For an MCB of a point set P , split \mathbf{b} into a set of sub-boxes $\{\mathbf{b}_1, \dots, \mathbf{b}_m\}$. Each of these sub-boxes is an MCB of a point set $P' \subset P$. Then let $B(c_{\mathbf{b}_i}, r_{\mathbf{b}_i})$ be the enclosing ball of sub-box $\mathbf{b}_i, 1 \leq i \leq m$. Define $rmax_{\mathbf{b}} = \max_i \{r_{\mathbf{b}_i}\}$. In case of $|\mathbf{b}| = 1$, let $rmax_{\mathbf{b}} = 0$.

Definition 6. Define $T_2(\mathbf{b}) = r_{\mathbf{b}} + (1 + 2/\epsilon)rmax_{\mathbf{b}}$.

Lemma 2. We have the following statements:

1. if $D(q, c_{\mathbf{b}}) \geq T_2(\mathbf{b})$, then every point in \mathbf{b} is ϵ -NN to q ;
2. if $rmax_{\mathbf{b}} < \frac{2}{2+\epsilon}r_{\mathbf{b}}$, then $T_2(\mathbf{b}) < T_1(\mathbf{b})$.

Proof. For the first statement, if $|\mathbf{b}| = 1$ then it is trivial. Assume $|\mathbf{b}| \geq 2$. Since the center of the enclosing ball of the box \mathbf{b} is chosen as any point in \mathbf{b} , it is easy to see that $c_{\mathbf{b}_i} \in \mathbf{b}$ for each sub-box \mathbf{b}_i . This in turn indicates that $D(c_{\mathbf{b}_i}, c_{\mathbf{b}}) \leq r_{\mathbf{b}}$. Thus, if $D(q, c_{\mathbf{b}}) \geq T_2(\mathbf{b})$, we have $D(q, c_{\mathbf{b}_i}) \geq D(q, c_{\mathbf{b}}) - D(c_{\mathbf{b}}, c_{\mathbf{b}_i}) \geq T_2(\mathbf{b}) - r_{\mathbf{b}} = (1 + 2/\epsilon)rmax_{\mathbf{b}} \geq (1 + 2/\epsilon)r_{\mathbf{b}_i} = T_1(\mathbf{b}_i)$. According to Lemma 1 we know that every point in \mathbf{b}_i is ϵ -NN to q . Since the subscript i is arbitrary in $[1, m]$, we conclude that every point in \mathbf{b} is ϵ -NN to q .

The second statement can be easily verified, and the proof is omitted here. \square

Situation 3 If q is still not as far from c_b as a distance of $T_2(\mathbf{b})$, it is time to ask the algorithm of (c, r) -NN for help. Let $\mathcal{A}(Q, \mathbf{q}, c, r)$ be any algorithm for solving (c, r) -NN, where Q is the input point set, \mathbf{q} is the query point, r is the query range, and c is the approximation factor. The meanings of these four parameters are already given in Definition 2. The goal of invoking \mathcal{A} is that, if \mathcal{A} answers *No* then still every point in \mathbf{b} is an ϵ -NN to q . The following lemma shows how to set the four input parameters to fulfill the goal.

Lemma 3. *Let $\mathcal{A}(Q, \mathbf{q}, c, r)$ be any algorithm for (c, r) -NN. We have the following statements:*

1. *if we set $Q = \{c_{\mathbf{b}_1}, \dots, c_{\mathbf{b}_m}\}$, $\mathbf{q} = q$, $r = \max_i \{T_2(\mathbf{b}_i)\}$, and let c satisfy $c \cdot r = \max_i \{T_1(\mathbf{b}_i)\}$, and invoke $\mathcal{A}(Q, \mathbf{q}, c, r)$, then if \mathcal{A} returns *No*, we can pick any point in \mathbf{b} as the answer of ϵ -NN to q ;*
2. *if $r \max_{\mathbf{b}_i} < \frac{2}{2+\epsilon} r_{\mathbf{b}_i}$ holds for each \mathbf{b}_i , $1 \leq i \leq m$, then our settings for c and r satisfy the requirement of (c, r) -NN problem definition. i.e. $c > 1$.*

Proof. For the first statement, according to Definition 2 of (c, r) -NN problem, if there exists a point in $\{c_{\mathbf{b}_1}, \dots, c_{\mathbf{b}_m}\}$ lying inside $B(q, r)$ where $r = \max_i \{T_2(\mathbf{b}_i)\}$, then \mathcal{A} will return some point $c_{\mathbf{b}_j}$ such that $c_{\mathbf{b}_j} \in B(q, cr)$ where $c \cdot r = \max_i \{T_1(\mathbf{b}_i)\}$. If all points in $\{c_{\mathbf{b}_1}, \dots, c_{\mathbf{b}_m}\}$ are outside $B(q, cr)$, then \mathcal{A} will return *No*. If the minimal distance from q to $\{c_{\mathbf{b}_1}, \dots, c_{\mathbf{b}_m}\}$ falls in the undefined range $[r, cr]$, \mathcal{A} will return either *No* or a point $c_{\mathbf{b}_j}$ such that $r \leq D(q, c_{\mathbf{b}_j}) \leq cr$.

On the other hand, according to Lemma 1 and 2, if all the points in $\{c_{\mathbf{b}_1}, \dots, c_{\mathbf{b}_m}\}$ satisfy $D(q, c_{\mathbf{b}_i}) \geq r = \max_i \{T_2(\mathbf{b}_i)\} \geq T_2(\mathbf{b}_i)$, then all points in all \mathbf{b}_i are ϵ -NN to q , $1 \leq i \leq m$, and these are already all points in \mathbf{b} .

Combining the two parts of analysis, we can conclude that if \mathcal{A} returns *No*, it must be the situation that the minimal distance from q to $\{c_{\mathbf{b}_1}, \dots, c_{\mathbf{b}_m}\}$ is not less than $r = \max_i \{T_2(\mathbf{b}_i)\}$. Equivalently, $D(q, c_{\mathbf{b}_i}) \geq \max_i \{T_2(\mathbf{b}_i)\} \geq T_2(\mathbf{b}_i)$ holds for each box \mathbf{b}_i , $1 \leq i \leq m$. Then according to Lemma 2, every point in \mathbf{b}_i is an ϵ -NN to q . Since the subscript i is arbitrary in $[1, m]$, we conclude that all points in \mathbf{b} are ϵ -NN to q .

For the second statement, if $r \max_{\mathbf{b}_i} < \frac{2}{2+\epsilon} r_{\mathbf{b}_i}$, then according to Lemma 2, $T_2(\mathbf{b}_i) < T_1(\mathbf{b}_i)$, $1 \leq i \leq m$. Taking maximum on both sides of the inequality, we have $\max_i \{T_2(\mathbf{b}_i)\} < \max_i \{T_1(\mathbf{b}_i)\}$. Since we set $r = \max_i \{T_2(\mathbf{b}_i)\}$, and let c satisfy $c \cdot r = \max_i \{T_1(\mathbf{b}_i)\}$, we have $r < cr$ which induces that $c > 1$. Then the proof is done. \square

Situation 4 As what is said in Lemma 3, if the algorithm \mathcal{A} returns *No* then the search of ϵ -NN terminates with returning an arbitrary point in \mathbf{b} . According to Definition 2, \mathcal{A} can also return some point $c_{\mathbf{b}_i} \in Q$ other than *No*. In that case the search must continue. At first glance, the same procedure should be recursively carried out, by applying Lemma 1, 2, 3 one by one on box \mathbf{b}_i , where the point $c_{\mathbf{b}_i}$ returned by \mathcal{A} is the center of the enclosing ball of box \mathbf{b}_i . However,

to guarantee that the algorithm returns a correct ϵ -NN, the box considered by the algorithm must enclose the exact NN p^* . But the box \mathbf{b}_i may not enclose p^* , which would ruin the correctness of the algorithm. Thus, we need to expand the search range to the boxes near to \mathbf{b}_i . The following Lemma 4 gives the bounds of the search range and ensures that p^* lies in the range.

Definition 7. For a collection of MCBs $\mathcal{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_m\}$, let $rmax_{\mathbf{b}_i}$ be defined as Definition 5 for each \mathbf{b}_i , $1 \leq i \leq m$. Then define $rmax_{\mathcal{B}} = \max_{\mathbf{b}_i \in \mathcal{B}} \{rmax_{\mathbf{b}_i}\}$.

Definition 8. Define $Nbr(\mathbf{b})$ as

$$\{\mathbf{b}' \in \mathcal{B} \mid D(c_{\mathbf{b}'}, c_{\mathbf{b}}) \leq (3 + 4\epsilon)rmax_{\mathcal{B}_{\mathbf{b}_s}}\}$$

where $\mathcal{B}_{\mathbf{b}_s} = Nbr(\mathbf{b}_s)$ and \mathbf{b}_s is the super box of \mathbf{b} .

Remark 4. The definition of Nbr sets is a recursive definition. For a box \mathbf{b} , its $Nbr(\mathbf{b})$ set is defined based on the $Nbr(\mathbf{b}_s)$ set of its super box \mathbf{b}_s . It requires that the boxes are recursively split, which can be represented as a tree structure. The formal description of the tree structure is given in Section 3.1.

Lemma 4. Given the query point q , and a collection of boxes $\{\mathbf{b}_1, \dots, \mathbf{b}_m\}$, if we find a box \mathbf{b}_i satisfying $D(q, c_{\mathbf{b}_i}) \leq \max_i \{T_1(\mathbf{b}_i)\}$, then the nearest neighbor of q lies in and can only lie in $Nbr(\mathbf{b}_i)$, i.e., $p^* \in Nbr(\mathbf{b}_i)$.

Proof. We have known that $D(q, c_{\mathbf{b}_i}) \leq \max_i \{T_1(\mathbf{b}_i)\}$. By the definition of the enclosing ball of box \mathbf{b} , the center of the ball is an arbitrary point picked from \mathbf{b} . Then we have $c_{\mathbf{b}_i} \in \mathbf{b}_i$ for box \mathbf{b}_i . Furthermore, since \mathbf{b}_i is a sub-box of \mathbf{b} , then $c_{\mathbf{b}_i} \in \mathbf{b}$ too. Thus $D(q, c_{\mathbf{b}_i})$ can serve as an upper bound of the distance from q to its nearest neighbor. Let p^* denote the exact nearest neighbor of q in \mathbf{b} , and we have $D(q, p^*) \leq D(q, c_{\mathbf{b}_i}) \leq \max_i \{T_1(\mathbf{b}_i)\}$. Further we have $\max_i \{T_1(\mathbf{b}_i)\} = \max_i \{(1 + 2/\epsilon)r_{\mathbf{b}_i}\} = (1 + 2/\epsilon)rmax_{\mathcal{B}}$. Then $D(q, p^*) \leq (1 + 2/\epsilon)rmax_{\mathcal{B}}$.

If p^* lies in some box \mathbf{b}_j , we prove $D(c_{\mathbf{b}_i}, c_{\mathbf{b}_j}) \leq (3 + 4/\epsilon)rmax_{\mathcal{B}}$. If $i = j$ then this trivially holds. If not, first we have $D(p^*, c_{\mathbf{b}_j}) \leq r_{\mathbf{b}_j} \leq rmax_{\mathcal{B}}$ since $p^* \in \mathbf{b}_j$. Thus, $D(c_{\mathbf{b}_i}, c_{\mathbf{b}_j}) \leq D(c_{\mathbf{b}_i}, q) + D(q, p^*) + D(p^*, c_{\mathbf{b}_j}) \leq (1 + 2/\epsilon)rmax_{\mathcal{B}} + (1 + 2/\epsilon)rmax_{\mathcal{B}} + rmax_{\mathcal{B}} = (3 + 4/\epsilon)rmax_{\mathcal{B}}$. This indicates that $\mathbf{b}_j \in Nbr(\mathbf{b}_i)$.

For a box \mathbf{b}_k out of $Nbr(\mathbf{b}_i)$, i.e. $D(c_{\mathbf{b}_k}, c_{\mathbf{b}_i}) > (3 + 4/\epsilon)rmax_{\mathcal{B}}$, suppose in contrary that the nearest neighbor $p^* \in \mathbf{b}_k$, which indicates $D(p^*, c_{\mathbf{b}_k}) \leq r_{\mathbf{b}_k}$. Then we have $D(q, p^*) \geq D(c_{\mathbf{b}_k}, c_{\mathbf{b}_i}) - D(c_{\mathbf{b}_i}, q) - D(c_{\mathbf{b}_k}, p^*) > (3 + 4/\epsilon)rmax_{\mathcal{B}} - (1 + 2/\epsilon)rmax_{\mathcal{B}} - r_{\mathbf{b}_k} > (1 + 2/\epsilon)rmax_{\mathcal{B}}$. This conflicts with the conclusion we get above.

So far both the sufficient and necessary conditions are proved, and the proof is done. \square

We are done introducing the mathematical preparations. In the next section we will propose our algorithm based on the lemmas given above.

3 Algorithms

In this section we propose our algorithm for reducing ϵ -NN to (c, r) -NN, including the preprocessing and query algorithm.

3.1 Preprocessing

Our preprocessing algorithm mainly consists of two sub-procedures. One is to build the box split tree T , and the other is to construct the Nbr sets.

Building the box split tree We first give the definition of the box split tree.

Definition 9 (Box split tree). *Given a point set P and its MCB \mathbf{b}_P , a tree T is a box split tree iff:*

1. *the root of T is \mathbf{b}_P ;*
2. *each non-root node of T is an MCB of a point set $P' \subset P$;*
3. *if box \mathbf{b}' is a sub-box of \mathbf{b} , then there is an edge between the node for \mathbf{b} and the node for \mathbf{b}' in T ;*
4. *each node has at least 2 child nodes, and at most $|P|$ child nodes;*
5. *$rmax_{\mathbf{b}} < \frac{2}{2+\epsilon}r_{\mathbf{b}}$ holds for each box \mathbf{b} in T .*

Further, T is fully built iff each box at the leaf nodes of T contains only one point.

Remark 5. The fifth term is required by the second statement of Lemma 3.

We use a box split method to build the box split tree. This method is originally proposed in [20], and also used in several other papers [11, 8]. It starts from the MCB \mathbf{b}_P of the point set P , then continuously splits \mathbf{b}_P into a collection \mathcal{B} of cubical boxes until each box in \mathcal{B} contains only one point. The whole produce of the method takes $O(dn \log n)$ time where $n = |P|$. The method proceeds in a series of split steps. In each split step, the box \mathbf{b}_L with the largest side length in the current collection \mathcal{B} is chosen and split. Define $h_i(\mathbf{b})$ to be the hyperplane orthogonal to the i -th coordinate axis and passing through the center of \mathbf{b} . One split step will split \mathbf{b}_L into at most 2^d sub-boxes using all $h_i(\mathbf{b}_L)$, each of which is an MCB. The set of non-empty sub-boxes generated by conducting one split step on \mathbf{b} is denoted as $Succ(\mathbf{b})$. The details of the box splitting method can be found in [20].

Next we describe how to use this method to build the box split tree T . The main obstacle is to satisfy the fifth term in Definition 9, i.e., $rmax_{\mathbf{b}} < \frac{2}{2+\epsilon}r_{\mathbf{b}}$ for each box \mathbf{b} in T . We use the following techniques to solve this problem.

When a split step is executed and a box \mathbf{b} is split, we temporarily store the sub-boxes of \mathbf{b} in a max-heap $H_{\mathbf{b}}$, which is ordered on the side length of the boxes in the heap. Recall the definitions in Section 2.2, the side length of a box \mathbf{b} is denoted as $len(\mathbf{b})$. When box \mathbf{b} is split fine enough so that $rmax_{\mathbf{b}} < \frac{2}{2+\epsilon}r_{\mathbf{b}}$ is satisfied, the algorithm will create a node for each $\mathbf{b}' \in H_{\mathbf{b}}$, and hang it under

the node for box \mathbf{b} in the box split tree T . Then for each \mathbf{b}' at these newly created leaf nodes, a max-heap is created to store its sub-boxes. In an overview, a max-heap is maintained for each box at the leaf nodes of the box split tree.

In each split step, the box with the largest volume is split. To efficiently pick out this box, a secondary heap H_2 is maintained. The heaps for the leaf nodes are called the primary heaps in contrast. The elements in H_2 is just the top elements in each primary heap, together with a pointer to its corresponding primary heap. Apparently the top element \mathbf{b}_{top} in H_2 is the box with largest volume. When \mathbf{b}_{top} is picked, the primary and secondary heap will pop it out simultaneously. Then \mathbf{b}_{top} is split by conducting one split step on it, generating $Succ(\mathbf{b})$. These sub-boxes in $Succ(\mathbf{b})$ will be added into the primary heap where \mathbf{b}_{top} formerly resides. When this primary heap finishes maintaining, its top element is inserted into the secondary heap. And then the iteration continues.

We point out the last problem to solve in order to satisfy the fifth term in Definition 9. The heaps, including the primary heaps and the secondary heap, are organized according to the len value of the boxes, in order to retrieve the box with the largest volume. On the other hand, the condition of $rmax_{\mathbf{b}} < \frac{2}{2+\epsilon}r_{\mathbf{b}}$ is based on the Est value of the boxes, because here we have $rmax_{\mathbf{b}} = \max_{\mathbf{b}' \in H_{\mathbf{b}}} \{Est(\mathbf{b}')\}$. Notice that the top element \mathbf{b}_{top} in the primary heap have the largest len value, but may not has the largest Est value. So we can not directly check $r_{\mathbf{b}_{top}} < \frac{2}{2+\epsilon}r_{\mathbf{b}}$ to decide whether \mathbf{b} is split fine enough. Fortunately, the len and Est value of a box have certain quantity relationships, which is formalized into the following lemma.

Lemma 5. *For the MCB \mathbf{b} of any point set P where $|\mathbf{b}| \geq 2$, we have $len(\mathbf{b}) \leq Est(\mathbf{b}) \leq d \cdot len(\mathbf{b})$. In the situation that $|\mathbf{b}| = 1$, we redefine $len(\mathbf{b})$ as $len(\mathbf{b}) = Est(\mathbf{b})$ to make this inequality consistent.*

Proof. The lemma already fixes the situation of $|\mathbf{b}| = 1$, and thus the proof focuses on when $|\mathbf{b}| \geq 2$.

If $len(\mathbf{b}) > Est(\mathbf{b})$, then \mathbf{b} can be shrunk and still contain all points in \mathbf{b} , which conflicts with that \mathbf{b} is the Minimal Cubical Box (MCB) of P . Thus $len(\mathbf{b}) \leq Est(\mathbf{b})$.

Recalling Equation 1, when $|\mathbf{b}| \geq 2$, $Est(\mathbf{b})$ is defined to be $D_{max}(\mathbf{b})$. Note that the L_q distance between two points in d -dimensional space is bounded by d times of the L_{∞} distance between them. Since the points are enclosed by box \mathbf{b} whose side length is $len(\mathbf{b})$, we conclude that $Est(\mathbf{b}) \leq d \cdot len(\mathbf{b})$.

Both sides of the inequality is proved. \square

With the help of Lemma 5, we have the following Lemma 6 about the criteria for deciding whether a box is split fine enough.

Lemma 6. *For box \mathbf{b} and its primary heap $H_{\mathbf{b}}$, if the top element \mathbf{b}_{top} satisfies $len(\mathbf{b}_{top}) < \frac{2}{(2+\epsilon)d}len(\mathbf{b})$, then $rmax_{\mathbf{b}} < \frac{2}{2+\epsilon}r_{\mathbf{b}}$.*

Proof. It's sufficient to prove $\forall \mathbf{b}' \in H_{\mathbf{b}}, Est(\mathbf{b}') < \frac{2}{2+\epsilon}Est(\mathbf{b})$.

Based on Lemma 5, we have $Est(\mathbf{b}') \leq d \cdot len(\mathbf{b}') \leq d \cdot len(\mathbf{b}_{top})$ for any box $\mathbf{b}' \in H_{\mathbf{b}}$. Combining with the condition given in the lemma, we have $Est(\mathbf{b}') < d \frac{2}{2+\epsilon} len(\mathbf{b}) = \frac{2}{2+\epsilon} len(\mathbf{b})$. And further combining with $len(\mathbf{b}) \leq Est(\mathbf{b})$, we finally have $Est(\mathbf{b}') < \frac{2}{2+\epsilon} Est(\mathbf{b})$, which proves the lemma. \square

The pseudo codes for building the box split tree are given in Algorithm 1. The algorithm also includes the invocation of Algorithm 2, aimed to maintain the Nbr sets, which will be introduced in the next section.

Algorithm 1: Preprocessing

Input: a point set P , and an approximation factor ϵ
Output: a box split tree T

// Initialization

- 1 Compute $\mathbf{b}_0 = MCB(P)$;
- 2 Compute the enclosing ball $B(c_{\mathbf{b}_0}, r_{\mathbf{b}_0})$ of \mathbf{b}_0 ;
- 3 Set \mathbf{b}_0 to be the root of T ;
- 4 Initialize the primary heap for \mathbf{b}_0 with one key-value pair $(len(\mathbf{b}_0), \mathbf{b}_0)$;
- 5 Initialize the secondary heap H_2 with one key-value pair $(len(\mathbf{b}_0), \mathbf{b}_0)$;

// Main loop

- 6 **while** $|\mathcal{B}| < n$ **do**
- 7 Pop out the top element \mathbf{b}_{top} from H_2 and its corresponding primary heap $H_{\mathbf{b}_s}$;
- 8 Split \mathbf{b}_{top} by conducting one split step on \mathbf{b} , generating $Succ(\mathbf{b}_{top})$;
- 9 **foreach** $\mathbf{b} \in Succ(\mathbf{b}_{top})$ **do**
- 10 Add \mathbf{b} into $H_{\mathbf{b}_s}$, and maintain the heap;
- 11 **end**
- 12 Let the current top element of $H_{\mathbf{b}_s}$ to be \mathbf{b}_t ;
- 13 Let $Flag = false$;
- 14 **if** $len(\mathbf{b}_t) < \frac{2}{(2+\epsilon)d} len(\mathbf{b}_s)$ **then** // Applying Lemma 6
- 15 Let $Flag = true$;
- 16 **foreach** $\mathbf{b} \in H_{\mathbf{b}_s}$ **do**
- 17 Create a node and hang it under the node of \mathbf{b}_s ;
- 18 Initialize the primary heap for \mathbf{b} with one key-value pair $(len(\mathbf{b}), \mathbf{b})$;
- 19 **end**
- 20 **else**
- 21 Add \mathbf{b}_t into H_2 .
- 22 **end**
- 23 Invoke Algorithm 2, taking $\mathbf{b}, Succ(\mathbf{b}), rmax_{\mathcal{B}_{\mathbf{b}_s}}$, and the boolean value $Flag$ as the input of this invocation;
- 24 **end**

Nbr sets maintaining Algorithm 2 for maintaining $Nbr(\mathbf{b})$ is given below. It is invoked each time the main loop of Algorithm 1 is executed, as shown above.

Algorithm 2: Maintaining $Nbr(b)$

Input: box \mathbf{b} , $Succ(\mathbf{b})$, the neighbor range parameter $rmax_{\mathcal{B}_{\mathbf{b}_s}}$, and a boolean value $Flag$.

```

1 foreach  $\mathbf{b}' \in Succ(\mathbf{b})$  do
2    $Nbr(\mathbf{b}') \leftarrow Nbr(\mathbf{b}) \cup Succ(\mathbf{b}) - \{\mathbf{b}\};$ 
3   Set  $Est(\mathbf{b}')$  according to Equation 1;
4   Update  $rmax_{\mathbf{b}_s}$ ;
5 end
6 foreach  $\mathbf{b}' \in Nbr(\mathbf{b})$  do
7   if  $Flag = true$  and  $\mathbf{b}'$  is in a higher level than  $\mathbf{b}$  then
8      $Nbr(\mathbf{b}') \leftarrow Nbr(\mathbf{b}') \cup Succ(\mathbf{b})$ 
9   else
10     $Nbr(\mathbf{b}') \leftarrow Nbr(\mathbf{b}') \cup Succ(\mathbf{b}) - \{\mathbf{b}\}$ 
11  end
12 end
13 foreach  $\mathbf{b}' \in Succ(\mathbf{b})$  do
14   foreach  $\mathbf{b}'' \in Nbr(\mathbf{b}')$  do
15     if  $D(c_{\mathbf{b}''}, c_{\mathbf{b}'}) > (3 + 4/\epsilon)rmax_{\mathcal{B}_{\mathbf{b}_s}}$  then
16       Delete  $\mathbf{b}''$  from  $Nbr(\mathbf{b}')$ ;
17       Delete  $\mathbf{b}'$  from  $Nbr(\mathbf{b}'')$ ;
18     end
19   end
20 end
```

There are two parts of Algorithm 2 that need to be explained in detail.

The first is Line 4. From Definition 8 for $Nbr(\mathbf{b})$, we can see that the maintaining of $Nbr(\mathbf{b})$ is based on the value $rmax_{\mathcal{B}_{\mathbf{b}_s}}$ passed down by its super-box \mathbf{b}_s . In Algorithm 2, Line 4 is aimed for updating $rmax_{\mathbf{b}_s}$ when the set of sub-boxes of \mathbf{b}_s is changed. If $Nbr(\mathbf{b})$ is implemented as a heap, then whenever any sub-box of \mathbf{b}_s needs $rmax_{\mathcal{B}_{\mathbf{b}_s}}$, this value can be retrieved from the heap in constant time.

The other part is the second *foreach* loop in Algorithm 2. The functionality of the loop is explained in the following Lemma 7.

Lemma 7. *The second foreach loop ensures that for all box \mathbf{b} in the box split tree T , each $\mathbf{b}' \in Nbr(\mathbf{b})$ is either in the same level with \mathbf{b} , or a degenerated box containing only one point.*

Proof. From Algorithm 1, we know that the boolean value $Flag$ indicates whether splitting \mathbf{b} causes the box split tree T to grow. If $Flag = true$, \mathbf{b} becomes an inner node. In that case, if there is a box $\mathbf{b}' \in Nbr(\mathbf{b})$ where \mathbf{b}' is in a higher level than \mathbf{b} , \mathbf{b} will remain in $Nbr(\mathbf{b}')$ according to the second *foreach* loop in Algorithm 2. First we claim that \mathbf{b}' is not an inner node. If so, \mathbf{b}' must have been split before, and Algorithm 2 was invoked at the time \mathbf{b}' was split. In this invocation, the *else*-branch of the second *foreach* loop was executed, and \mathbf{b}' was already deleted

from $Nbr(\mathbf{b})$. This conflicts with $\mathbf{b}' \in Nbr(\mathbf{b})$. Thus, we get the conclusion that \mathbf{b}' is not an inner node and never has been split.

We count on the next several invocations of Algorithm 2 to delete \mathbf{b}' from $Nbr(\mathbf{b})$. After \mathbf{b} is split, \mathbf{b}' may be split but the boxes in $Succ(\mathbf{b}')$ may be in the same level with \mathbf{b}' , which only introduces more higher-level boxes into $Nbr(\mathbf{b})$. The critical time is when $Succ(\mathbf{b}')$ is in the next level of \mathbf{b}' . In that case, While Algorithm 2 is invoked by splitting \mathbf{b}' , the *else*-branch of the second *foreach* loop will delete \mathbf{b}' from $Nbr(\mathbf{b})$. Of course, if \mathbf{b}' contains only one point and can not be split, it will remain in $Nbr(\mathbf{b})$ until Algorithm 1 terminates.

So far we have eliminated a box in $Nbr(\mathbf{b})$ containing more than one point and in a higher level than \mathbf{b} . Repeatedly applying the same proof, we will eliminate all such box in \mathbf{b} . Then the claim is proved. \square

3.2 Query

The query algorithm goes down the tree T returned by Algorithm 1 level by level. At each level of T , the algorithm \mathcal{A} for (c, r) -NN will be invoked, and the input parameters of \mathcal{A} are set according to Lemma 3. The pseudo codes are given in Algorithm 3.

We should spend some efforts to explain the termination condition in Algorithm 3. First we introduce a lemma about $Nbr(\mathbf{b})$ when $|\mathbf{b}| = 1$.

Lemma 8. *For a box \mathbf{b} satisfying $|\mathbf{b}| = 1$, all the boxes $\mathbf{b}' \in Nbr(\mathbf{b})$ contain only one point, i.e., $|\mathbf{b}'| = 1$.*

Proof. According to Algorithm 2, if a box \mathbf{b} satisfies $|\mathbf{b}| = 1$, the algorithm will keep updating $Nbr(\mathbf{b})$ until Algorithm 2 is not invoked any more. And that is when Algorithm 1 terminates and when all the boxes degenerate and contain only one point. It implies that any box $\mathbf{b}' \in Nbr(\mathbf{b})$ satisfies $|\mathbf{b}'| = 1$. \square

According to the above lemma, when the WHILE loop breaks, all boxes in $\mathbf{b}_c \cup Nbr(\mathbf{b}_c)$ contains only one point. The brute-force search takes $O(|Nbr(\mathbf{b}_c)|)$ time. We will bound this complexity in the next section.

4 Analysis

4.1 Correctness

First we prove the correctness of our query algorithm by introduce the following lemma 9.

Lemma 9. *In every execution of the loop body, Algorithm 3 ensures that the exact nearest neighbor $p^* \in P_c$ after the assignment of P_c (Line 8).*

Proof. The proof proceeds by induction. At the beginning of Algorithm 3, apparently we have $P_c = P$, and $p^* \in P_c$ holds trivially. As inductive hypothesis, we assume $p^* \in P_c$ after Line 8 in one execution of the loop body. Consider the rest of

Algorithm 3: Query

Input: query point q , data set P , box split tree T , and algorithm \mathcal{A} for (c, r) -NN

Output: ϵ -NN of q in P

```
1 set  $\mathbf{b}_c = \text{root}(T)$ ;  
2 if  $D(q, c_{\mathbf{b}_c} \geq T_2(\mathbf{b}))$  then  
3   pick any point  $p' \in \mathbf{b}_c \cap P$ ;  
4   return  $p'$ ;  
5 end  
6 while  $|\mathbf{b}_c| > 1$  do  
7    $\mathcal{B}_c \leftarrow \text{Nbr}(\mathbf{b}_c)$ ;  
8    $P_c \leftarrow \bigcup_{\mathbf{b} \in \mathcal{B}_c} \mathbf{b} \cap P$ ;  
9   invoke  $\mathcal{A}$ , where the input of  $\mathcal{A}$  is set according to Lemma 3;  
10  if the query returns NO then  
11    pick any point  $p' \in P_c$ ;  
12    return  $p'$ ;  
13  else // the query returns the center  $c_{\mathbf{b}'}$  of box  $\mathbf{b}'$   
14    set  $\mathbf{b}_c = \mathbf{b}'$ ;  
15    continue;  
16  end  
17 end  
18  $P_c \leftarrow \text{Nbr}(\mathbf{b}_c) \cap P$ ;  
19 Conduct brute-force search in  $P_c$  to find the exact NN;
```

the loop body. If Line 12 is executed, the algorithm will return, and the induction finishes. If Line 14 is executed, Lemma 4 ensures that $p^* \in \text{Nbr}(\mathbf{b}') = \text{Nbr}(\mathbf{b}_c)$. Thus, in the next execution of the loop body, the reassignment of P_c at Line 8 makes $p^* \in P_c$ to hold again. Then by mathematical induction, the lemma is proved. \square

Theorem 1 (Correctness). *The point p' returned by Algorithm 3 is an ϵ -NN to q in P , i.e., if p^* is the exact NN to q in P , then $D(q, p') \leq (1 + \epsilon)D(q, p^*)$.*

Proof. Considering Algorithm 3, if it returns at 4, Lemma 2 ensures that the picked point p' is an ϵ -NN to q ; if it returns at Line 12, Lemma 3 ensures that the point p' returned here is an ϵ -NN to q ; and if the algorithm finally goes out of the WHILE loop and executes brute force search in the final P_c assigned at Line 18, Lemma 9 ensures that the exact nearest neighbor lies in P_c , and thus the brute-force search returns the exact nearest neighbor to q for sure. \square

4.2 Complexities

Before we bound the complexity of our algorithm, we should first bound the size of $\text{Nbr}(\mathbf{b})$ for any box \mathbf{b} by introducing a lemma from [20].

Lemma 10 ([20]). *Let r be a positive number. During the execution of the split method described in Section 3.1, at each time before splitting a box, let \mathcal{B} be the current box collection, and let \mathbf{b}_L be the box with the largest volume in \mathcal{B} . For any box $\mathbf{b} \in \mathcal{B}$, the size of the set $\{\mathbf{b}' \in \mathcal{B} \mid D_{\min}(\mathbf{b}, \mathbf{b}') \leq r \cdot \text{Est}(\mathbf{b}_L)\}$ is at most $2^d(2d\lceil r \rceil + 3)^d$.*

Based on the lemma above, we can bound the size of $Nbr(\mathbf{b})$ for any box \mathbf{b} in the box split tree T constructed in Algorithm 1.

Lemma 11. *The size of $Nbr(\mathbf{b})$ defined in Definition 8 and constructed in Algorithm 2 is $O((\frac{d}{\epsilon})^d)$.*

Proof. We prove this by using Lemma 10.

By Definition 8, $Nbr(\mathbf{b}) = \{\mathbf{b}' \mid D(c_{\mathbf{b}'}, c_{\mathbf{b}}) \leq (3 + 4\epsilon)rmax_{\mathcal{B}_{\mathbf{b}_s}}\}$, where $\mathcal{B}_{\mathbf{b}_s} = Nbr(\mathbf{b}_s)$ and \mathbf{b}_s is the super box of \mathbf{b} . On the other hand, Lemma 10 concerns the set $\{\mathbf{b}' \in \mathcal{B} \mid D_{\min}(\mathbf{b}', \mathbf{b}) \leq r \cdot \text{Est}(\mathbf{b}_L)\}$, where \mathbf{b}_L is the box with the largest volume in the box collection \mathcal{B} . We should fill the gap between $Nbr(\mathbf{b})$ and $\{\mathbf{b}' \in \mathcal{B}' \mid D_{\min}(\mathbf{b}', \mathbf{b}) \leq r \cdot \text{Est}(\mathbf{b}_L)\}$ by considering two relationships: 1) $D(c_{\mathbf{b}'}, c_{\mathbf{b}})$ and $D_{\min}(\mathbf{b}', \mathbf{b})$, and 2) $rmax_{\mathcal{B}_{\mathbf{b}_s}}$ and $\text{Est}(\mathbf{b}_L)$.

1) Since the center of the enclosing ball of box \mathbf{b} is one arbitrary point inside \mathbf{b} , we have $D_{\min}(\mathbf{b}, \mathbf{b}') \leq D(c_{\mathbf{b}}, c_{\mathbf{b}'})$. Thus, it can be easily verified that:

$$\forall K > 0, \{\mathbf{b}' \mid D(c_{\mathbf{b}}, c_{\mathbf{b}'}) \leq K\} \subseteq \{\mathbf{b}' \mid D_{\min}(\mathbf{b}, \mathbf{b}') \leq K\}$$

2) Recall Definition 7, $rmax_{\mathcal{B}_{\mathbf{b}_s}} = \max_{\mathbf{b}' \in \mathcal{B}_{\mathbf{b}_s}} rmax_{\mathbf{b}'} = \max_{\mathbf{b}' \in \mathcal{B}_{\mathbf{b}_s}} \max_{\mathbf{b}'' \in Chd(\mathbf{b}')} \text{Est}(\mathbf{b}'')$ where $Chd(\mathbf{b}')$ is the set of sub-boxes of \mathbf{b}' . Let $\mathcal{B}' = \bigcup_{\mathbf{b}' \in \mathcal{B}_{\mathbf{b}_s}} Chd(\mathbf{b}')$, and \mathcal{B}' is clearly a subset of the whole box collection \mathcal{B} . On the other hand, \mathbf{b}_L is the box with the largest volume in \mathcal{B} . Based on Fact 5, we have $rmax_{\mathcal{B}_{\mathbf{b}_s}} = \max_{\mathbf{b}' \in \mathcal{B}'} \{\text{Est}(\mathbf{b}')\} \leq \max_{\mathbf{b}' \in \mathcal{B}'} \{d \cdot \text{len}(\mathbf{b}')\} \leq d \cdot \text{len}(\mathbf{b}_L) \leq d \cdot \text{Est}(\mathbf{b}_L)$. Thus we have

$$\forall \alpha > 0, \{\mathbf{b}' \mid D_{\min}(\mathbf{b}, \mathbf{b}') \leq \alpha \cdot rmax_{\mathcal{B}_{\mathbf{b}_s}}\} \subseteq \{\mathbf{b}' \mid D_{\min}(\mathbf{b}, \mathbf{b}') \leq \alpha \cdot d \cdot \text{Est}(\mathbf{b}_L)\}$$

Combining 1) and 2), we have:

$$\{\mathbf{b}' \mid D(c_{\mathbf{b}}, c_{\mathbf{b}'}) \leq (3 + 4/\epsilon)rmax_{\mathcal{B}_{\mathbf{b}_s}}\} \subseteq \{\mathbf{b}' \mid D_{\min}(\mathbf{b}, \mathbf{b}') \leq (3 + 4/\epsilon)d \cdot \text{Est}(\mathbf{b}_L)\}$$

Then according to Lemma 10, we have

$$|\{\mathbf{b}' \mid D_{\min}(\mathbf{b}, \mathbf{b}') \leq (3 + 4/\epsilon)d \cdot \text{Est}(\mathbf{b}_L)\}| \leq 2^d(2d\lceil (3 + 4/\epsilon)d \rceil + 3)^d = O((\frac{d}{\epsilon})^d)$$

Note that $Nbr(\mathbf{b}_s)$ may be updated and the value of $rmax_{\mathcal{B}_{\mathbf{b}_s}}$ may be changed while $Nbr(\mathbf{b})$ is being maintained based on an older value of $rmax_{\mathcal{B}_{\mathbf{b}_s}}$. But it does not influent the result in this lemma, because Lemma 10 ensures that the size of the set considered in the lemma is bounded *every time* before a box is split. Thus, even though $Nbr(\mathbf{b})$ may be maintained based on an older value of $rmax_{\mathcal{B}_{\mathbf{b}_s}}$, $|Nbr(\mathbf{b})| = O((\frac{d}{\epsilon})^d)$ still holds. \square

We introduce and prove another lemma which is about the property of the box split tree T constructed in preprocessing phase.

Lemma 12. *For a point set P where $|P| = n$, the fully built split tree T constructed based on P has the following properties:*

1. *There are at most $2n$ nodes in T .*
2. *The total time to build T is $O(dn \log n)$.*

Proof. For the first statement, the proof starts from the following two observations.

- 1) There are exactly n leaf nodes in T . Because T is fully built, each box at leaf node contains only one point.
- 2) Each node has at least 2 child nodes and at most $|P| = n$ child nodes. This comes from the definition of box split tree.

Combining the two observations, if T is a full binary tree, then T has at most $2n$ nodes, which can be easily verified. As long as one node has three child nodes or more, the total number of nodes would be less than $2n$. The extreme situation is that the root has n child nodes and there are totally $n + 1$ nodes in T . So we can conclude that there are at most $2n$ nodes in T .

For the second statement, we can divide the time to build T into two parts. One is the total time to conduct all the split steps. The other is the total time to manipulate the primary and secondary heaps. We analyze the time complexity of the two parts as follows.

- 1) The total time to conduct the split steps is $O(dn \log n)$. This is already proved in [20]. We omit the proof and refer the readers to [20] for the details.
- 2) We have proved that there are at most $2n$ boxes in the fully built split tree T . Considering the manipulation of the primary and secondary heaps, it is easily verified that each box \mathbf{b} may exist in at most two heaps, i.e., one primary and one secondary. For each heap, \mathbf{b} can only be pushed into it only once, and be popped out of it only once. The number of the boxes in the heap is at most $2n$, so for each box \mathbf{b} , the heap manipulation time incurred by \mathbf{b} is $O(\log n)$. Thus, the total time to manipulate the primary and secondary heaps is $O(n \log n)$.

Adding the two parts of complexity, we conclude that the total time to build T is $O(dn \log n)$. \square

Now we start to prove the complexities of our algorithm, including preprocessing time, space and query time complexities.

Theorem 2 (Preprocessing Time Complexity). *The complexity of Algorithm 1 for preprocessing is $O((\frac{d}{\epsilon})^d \cdot n \log n)$.*

Proof. The complexity of Algorithm 1 can be divided into two parts, namely, (1) the total time to build the box split tree T , and (2) the total time to maintain Nbr data structures and Est value for all boxes. The first part of complexity is already proved in Lemma 12. Here we prove the second part.

Our Algorithm 2 is very similar to an algorithm for maintaining the Nbr set in [20]. We prove the complexity of Algorithm 1 by similar techniques in [20]. If

the $Nbr(\mathbf{b})$ sets is implemented by a heap, which allows insertion and deletion in $O(\log n)$ time, and allow access to largest value of $D_{min}(\mathbf{b}, \mathbf{b}'), \mathbf{b}' \in Nbr(\mathbf{b})$ in $O(\log n)$ time, then we can use the similar analysis in [20] and bound the time to maintain Nbr sets and Est values by $O((\frac{d}{\epsilon})^d \cdot n \log n)$. The details are omitted.

In summary, we have proved the desired preprocessing complexity. \square

Theorem 3 (Space Complexity). *The space complexity of Algorithm 1 is $O((\frac{d}{\epsilon})^d \cdot n)$.*

Proof. The space complexity of the algorithm is bounded by the number of boxes in the tree T multiplying the size of $Nbr(\mathbf{b})$ sets maintained for each box. According to Lemma 12, there are at most $2n$ boxes in T . And according to Lemma 11, $|Nbr(\mathbf{b})| = O((\frac{d}{\epsilon})^d)$. Multiplying the two factors, we get the desired result. \square

Theorem 4 (Query Time Complexity). *Algorithm 3 invokes $O(\log n)$ times of the algorithm \mathcal{A} for (c, r) -NN problem.*

Proof. Considering the box split tree returned by Algorithm 1, its fan-out, i.e., the number of child nodes of a node, is at least 2. This comes from the definition of box split tree. And the number of leaf nodes is n , so that the height of the tree is $O(\log n)$. Further, by Lemma 7 we know that all boxes in any box \mathbf{b} in T are in the same level with \mathbf{b} . Thus, Algorithm 3 invokes at most one (c, r) -NN query at each level of the tree. Hence, the number of invoked (c, r) -NN queries is $O(\log n)$. \square

5 Conclusion

In this paper we proposed a new algorithm for reducing ϵ -NN problem to (c, r) -NN problem. Compared to the former works for the same reduction problem, our algorithm achieves the lowest query time complexity, which is $O(\log n)$ times of invocations of the algorithm for (c, r) -NN problem. We elaborately designed the input parameters of each of the invocation, and built a dedicated data structure in preprocessing phase to support the query procedure. A box split method proposed in [20] is used as a building block for the algorithm of preprocessing phase. Our paper also raises a problem which is to reduce the exponential complexity on d introduced by the box split method. This is left as our future work.

References

1. Andoni, A., Indyk, P.: Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In: 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). vol. 51, pp. 459–468. IEEE (2006)
2. Andoni, A., Indyk, P.: Nearest Neighbors In High-Dimensional Spaces. In: Handbook of Discrete and Computational Geometry, chap. 43, pp. 1135–1155. CRC Press, Inc, 3rd edn. (2017)

3. Andoni, A., Indyk, P., Nguyn, H.L., Razenshteyn, I.: Beyond Locality-Sensitive Hashing. In: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1018–1028. No. 1, Society for Industrial and Applied Mathematics, Philadelphia, PA (jan 2014)
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 '15. pp. 793–801. ACM Press, New York, New York, USA (2015)
5. Arya, S., Mount, D.M.: Approximate Nearest Neighbor Queries in Fixed Dimensions. In: Proceedings of the Fourth Annual {ACM/SIGACT-SIAM} Symposium on Discrete Algorithms. pp. 271–280 (dec 1993)
6. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM* **45**(6), 891–923 (nov 1998)
7. Bern, M.W.: Approximate Closest-Point Queries in High Dimensions. *Inf. Process. Lett.* **45**(2), 95–99 (1993)
8. Callahan, P.B., Kosaraju, S.R.: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM* **42**(1), 67–90 (jan 1995)
9. Chan, T.M.: Approximate nearest neighbor queries revisited. In: Proceedings of the thirteenth annual symposium on Computational geometry - SCG '97. vol. 20, pp. 352–358. ACM Press, New York, New York, USA (1997)
10. 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 '04. p. 253. ACM Press, New York, New York, USA (2004)
11. Feder, T., Greene, D.: Optimal algorithms for approximate clustering. In: Proceedings of the twentieth annual ACM symposium on Theory of computing - STOC '88. pp. 434–444. ACM Press, New York, New York, USA (1988)
12. Goel, A., Indyk, P., Varadarajan, K.: Reductions Among High Dimensional Proximity Problems. In: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 769–778. Society for Industrial and Applied Mathematics (2001)
13. Har-Peled, S.: A replacement for Voronoi diagrams of near linear size. In: Proceedings 2001 IEEE International Conference on Cluster Computing. pp. 94–103. IEEE Comput. Soc (2001)
14. Har-Peled, S., Indyk, P., Motwani, R.: Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory of Computing* **8**(1), 321–350 (2012)
15. 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 '98. pp. 604–613. ACM Press, New York, New York, USA (1998)
16. Kleinberg, J.M.: Two algorithms for nearest-neighbor search in high dimensions. In: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97. pp. 599–608. ACM Press, New York, New York, USA (1997)
17. Krauthgamer, R., Lee, J.R.: Navigating nets: simple algorithms for proximity search. In: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 798–807 (2004)
18. Kushilevitz, E., Ostrovsky, R., Rabani, Y.: Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. *SIAM Journal on Computing* **30**(2), 457–474 (jan 2000)

19. Panigrahy, R.: Entropy based nearest neighbor search in high dimensions. In: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm - SODA '06. pp. 1186–1195. ACM Press, New York, New York, USA (2006)
20. Vaidya, P.M.: An optimal algorithm for the all-nearest-neighbors problem. In: Intergovernmental Panel on Climate Change (ed.) 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). pp. 117–122. IEEE, Cambridge (oct 1986)