CSD: Discriminance with Conic Section for Improving Reverse \( k \) Nearest Neighbors Queries

Yang Li\( ^\dagger \), Gang Liu\( ^\dagger \), Mingyuan Bai\( ^\dagger \), Junbin Gao\( ^\dagger \), Lixin Ye\( ^*\) and Zi Ming\( ^\dagger \)

\( ^*\)School of Computer Science, China University of Geosciences, Wuhan, China

\( ^\dagger \)Hubei Key Laboratory of Intelligent Geo-Information Processing, China University of Geosciences, Wuhan, China

\( ^\dagger \)The University of Sydney Business School, The University of Sydney, Sydney, NSW, Australia

\( ^\dagger \)School of Economics and Management, Hubei University of Technology, Wuhan, China

\{liyang_cs, liugang\}@cug.edu.cn, \{mbai8854@uni., junbin.gao@\}@sydney.edu.au, yelixin.cs@gmail.com, ziming@cug.edu.cn

Abstract—The reverse \( k \) nearest neighbor (R\( k \)NN) query finds all points that have the query point as one of their \( k \) nearest neighbors (kNN), where the kNN query finds the \( k \) closest points to its query point. Based on the characteristics of conic section, we propose a discrimination, named CSD (Conic Section Discriminance), to determine points whether belong to the R\( k \)NN set without issuing any queries with non-constant computational complexity. By using CSD, we also implement an efficient R\( k \)NN algorithm CSD-R\( k \)NN with a computational complexity at \( O(k^{1.3} \cdot \log k) \). The comparative experiments are conducted between CSD-R\( k \)NN and other two state-of-the-art R\( k \)NN algorithms, SLICE and VR-R\( k \)NN. The experimental results indicate that the efficiency of CSD-R\( k \)NN is significantly higher than its competitors.

Index Terms—R\( k \)NN, conic section, Voronoi, Delaunay

I. INTRODUCTION

As a variant of nearest neighbor (NN) query, RNN query is first introduced by Korn and Muthukrishnan \[1\]. A direct generalization of NN query is the reverse \( k \) nearest neighbors (R\( k \)NN) query, where all points having the query point as one of their \( k \) closest points are required to be found. Since its appearance, R\( k \)NN has received extensive attention \[2\], \[3\], \[4\], \[5\], \[6\] and \[7\] and been prominent in various scientific fields including machine learning, decision support, intelligent computation and geographic information systems, etc.

At first glance, R\( k \)NN and kNN queries appear to be equivalent, meaning that the results for R\( k \)NN and kNN may be the same for the same query point. However, R\( k \)NN is not as simple as it seems to be. It is a very different kind of query from kNN, although their results are similar in many cases. So far, R\( k \)NN is still an expensive query for its computational complexity at \( O(k^2) \) \[9\], whereas the computational complexity of kNN queries has been reduced to \( O(k \cdot \log k) \) \[7\].

In order to solve the RNN/R\( k \)NN problem, a large number of approaches have been proposed. Some early methods \[8\], \[1\], \[9\] speed up RNN/R\( k \)NN queries by pre-computation. Their disadvantage is that it is difficult to support queries on dynamic data sets. Therefore, many R\( k \)NN algorithms without pre-computation are proposed.

Most existing non-pre-computation R\( k \)NN algorithms have two phases: the filtering phase and the refining phase (also known as the pruning phase and the verification phase). In the pruning phase, the majority of points that do not belong to R\( k \)NN should be filtered out. The main goal of this phase is to generate a candidate set as small as possible. In the verification phase, each candidate point should be verified whether it belongs to the R\( k \)NN set or not. For most algorithms, the candidate points are verified by issuing kNN queries or range queries, which are very computational expensive. The state-of-the-art R\( k \)NN technique SLICE, provides a more efficient verification method with a computational complexity of \( O(k) \) for one candidate. The size of the candidate set of SLICE varies form \( 2k \) to \( 3.1k \). However, it is still time consuming to perform such a verification for each candidate point.

There seems to be a consensus in the past studies that for an R\( k \)NN technique, the number of verification points cannot be smaller than the size of the result set. Such an idea, however, limits our understanding of the R\( k \)NN problem. Hence we amend our thought and come up with a conjecture that whether a point could be directly determined as belonging to the R\( k \)NN set or not. For most algorithms, the verification phase, each candidate point should be verified whether it belongs to the R\( k \)NN set or not. For most algorithms, the candidate points are verified by issuing kNN queries or range queries, which are very computational expensive. The state-of-the-art R\( k \)NN technique SLICE, provides a more efficient verification method with a computational complexity of \( O(k) \) for one candidate. The size of the candidate set of SLICE varies form \( 2k \) to \( 3.1k \). However, it is still time consuming to perform such a verification for each candidate point.

There seems to be a consensus in the past studies that for an R\( k \)NN technique, the number of verification points cannot be smaller than the size of the result set. Such an idea, however, limits our understanding of the R\( k \)NN problem. Hence we amend our thought and come up with a conjecture that whether a point could be directly determined as belonging to the R\( k \)NN set or not. For most algorithms, the verification phase, each candidate point should be verified whether it belongs to the R\( k \)NN set or not. For most algorithms, the candidate points are verified by issuing kNN queries or range queries, which are very computational expensive. The state-of-the-art R\( k \)NN technique SLICE, provides a more efficient verification method with a computational complexity of \( O(k) \) for one candidate. The size of the candidate set of SLICE varies form \( 2k \) to \( 3.1k \). However, it is still time consuming to perform such a verification for each candidate point.

Table \[I\] shows the comparison of computational complexity among VR-R\( k \)NN, SLICE and CSD-R\( k \)NN. It can be seen that the bottleneck of both VR-R\( k \)NN and SLICE is the
reverse nearest neighbor (RNN) queries are first introduced by Korn and Muthukrishnan where RNN queries are implemented by preprocessing the database. For each point \( p \) in the database, a circle with \( p \) as the center and the distance from \( p \) to its nearest neighbor as the radius is pre-calculated and these circles are indexed by an R-tree. The RNN set of a query point \( q \) includes all the points whose circle contains \( q \). With the R-tree, the RNN set of any query point can be found efficiently. Soon after, several techniques \([10,11]\) are proposed to improve their work.

### B. Six-regions

Six-regions \([2]\) algorithm, proposed by Stanoï et al., is the first approach that does not need any pre-computation. They divide the space into six equal segments using six rays starting at the query point, so that the angle between the two boundary rays of each segment is \( 60^\circ \). They suggest that only the nearest neighbor (NN) of the query point in each of the six segments may belong to the RNN set. It firstly performs six NN queries to find the closest point of the query point \( q \) in each segments. Then it launches an NN query for each of the six points to verify \( q \) as their NN. Finally the RNN of \( q \) is obtained.

Generalizing this theory to R\( k \)-NN queries leads to a corollary that, only the members of \( k \)-NN of the query point in each segment have the possibility of belonging to the R\( k \)-NN set. This corollary is widely adopted in the pruning phase of several R\( k \)-NN techniques.

### C. TPL

TPL \([3]\), proposed by Tao et al., is one of the prestigious algorithms for R\( k \)-NN queries. This technique prunes the space using the bisectors between the query point and other points. The perpendicular bisector is denoted by \( B_{pq} \), \( B_{pq} \) divides the space into two half-spaces. The half-space that contains \( p \) is denoted as \( H_{pq} \). Another one is denoted as \( H_{qp} \). If a point \( p' \) lies in \( H_{pq} \), \( p' \) must be closer to \( p \) than to \( q \). Then \( p' \) cannot be the RNN of \( q \) and we can say that \( pq \) prunes \( p' \). If a point is pruned by at least \( k \) other points, then it cannot belong to the RNN of \( q \).

An area that is the intersection of any combination of \( k \) half-spaces can be pruned. The total pruned area corresponds to the union of pruned regions by all such possible combinations of \( k \) bisectors (total \( \binom{m}{k} \) combinations). TPL also uses an alternative computational cheaper pruning method which has a less pruning power. All the points are sorted by their Hilbert values. Only the combinations of \( k \) consecutive points are used to prune the space (total \( m \) combinations).

### D. FINCH

FINCH is another famous R\( k \)-NN algorithm proposed by Wu et al. \([4]\). The authors of FINCH think that it is too computational costly to use \( m \) combinations of \( k \) bisectors to prune the points. They utilize a convex polygon that approximates the unpinned region to prune the points instead of using bisectors. All points lying outside the polygon should be pruned. Since the containment can be achieved in logarithmic time for convex polygons, the pruning of FINCH has a higher efficiency than TPL. However, the computational complexity of computing the approximately unpinned convex polygon is \( O(m^3) \), where \( m \) is the number of points considered for pruning.

### E. InfZone

Previous techniques can reduce the candidate set to an extent by different pruning methods. However, their verification methods for candidates are very inefficient. It is quite computational costly to issue an inefficient verification for each point in a candidate set with a size of \( O(k) \). In order to overcome this issue, a novel R\( k \)-NN technique which is named as InfZone is proposed by Cheema et al. \([5]\). The authors of InfZone introduce the concept of influence zone (denoted as \( Z_k \)), which also can be called R\( k \)-NN region. The influence zone of a query point \( q \) is a region that, a point \( p \) belongs to the R\( k \)-NN set of \( q \), if and only if it lies in the \( Z_k \) of \( q \). The influence zone is always a star-shaped polygon and the query point is its kernel point. A number of properties are detailed. These properties are aimed to shrink the number of points which are crucial to compute the influence zone. They propose an influence zone computing algorithm with a computational complexity of \( O(k \cdot m^3) \), where \( m \) is the number of points accessed during the construction of the influence zone. Every points that lies inside the influence zone are accessed in the pruning phase, since they cannot be ignored during the
construction of the influence zone. Namely, all the potential members of the R$k$NN are accessed during the pruning phase. Hence, for monochromatic R$k$NN queries, InfZone does not require to verify the candidates. It is indicated that the expected size of R$k$NN set is $k$. Evidently, the size of R$k$NN must not be greater than $m$, i.e., $k \leq m$. Therefore, the computational complexity of InfZone must be no less than $O(k^3)$.

F. SLICE

SLICE [6] is the state-of-the-art approach for R$k$NN queries. In recent years, several well-known techniques [2] have been proposed to address the limitations of half-space pruning (e.g., FINCH [4], InfZone [5]). While few researchers carried out further research based on the idea of Six-regions. Yang et al. suggests that the regions-based pruning approach of Six-regions has great potential and proposed an efficient R$k$NN algorithm SLICE [6]. SLICE uses a more powerful and flexible pruning approach that prunes a much larger area as compared to Six-regions with almost similar computational complexity. Furthermore, it significantly improves the verification phase by computing a list of significant points for each segment. These lists are named as sigLists. Each candidate can be verified by accessing sigList instead of issuing a range query. Therefore, SLICE is significantly more efficient than the other existing algorithms.

G. VR-R$k$NN

For most R$k$NN algorithms, data points are indexed by R-tree [12]. However, R-tree is originally designed primarily for range queries. Although some approaches [13], [3], [14], [15] are proposed afterwards to make it also suitable for NN queries and their variants: the NN derived queries are still disadvantageous. When answering an NN derived query, all nodes in the R-tree intersecting with the local neighborhood (Search Region) of the query point need to be accessed to find all the members of the result set. Once the candidate set of the query is large, the cost of accessing the nodes can also become very large. In order to improve the performance of R-tree on NN derived queries, Sharifzadeh and Shahabi proposes a composite index structure composed of an R-tree and a Voronoi diagram, and named it as VoR-Tree [7]. VoR-Tree benefits from both the neighborhood exploration capability of Voronoi diagrams and the hierarchical structure of R-tree. By utilizing VoR-tree, they propose VR-R$k$NN to answer the R$k$NN query. Similar to the filter phase of Six-regions [4], VR-R$k$NN divides the space into 6 equal segments and selects $k$ candidate points from each segment to form a candidate set of size $6k$. During the refining phase, each candidate point is verified to be a member of the R$k$NN through issuing a kNN query (VR-kNN). The expected computational complexity of VR-R$k$NN is $O(k^2 \cdot \log k)$

III. PRELIMINARIES

First, we formally define the problem in Section 3.1. Then we present some concepts and knowledge to enhance the understanding of our methodologies in Section 3.2 and Section 3.3.

A. Problem definition

Definition 1. Euclidean Distance: Given two points $A = \{a_1, a_2, ..., a_d\}$ and $B = \{b_1, b_2, ..., b_d\}$ in $\mathbb{R}^d$, the Euclidean distance between $A$ and $B$, $dist(A, B)$, is defined as follows:

$$
    dist(A, B) = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2}.
$$

(1)

Definition 2. kNN Queries: A kNN query is to find the $k$ closest points to the query point from a certain point set. Mathematically, this query in Euclidean space can be stated as follows. Given a set $P$ of points in $\mathbb{R}^d$ and a query point $q \in \mathbb{R}^d$,

$$
    kNN(q) = \{p \in P \mid dist(p, q) \leq dist(p_k, q)\}
$$

where $p_k$ is the $k$th closest point to $q$ in $P$.

(2)

Definition 3. R$k$NN Queries: A R$k$NN query retrieves all the points that have the query point as one of their $k$ nearest neighbors from a certain point set. Formally, given a set $P$ of points in $\mathbb{R}^d$ and a query point $q \in P$, the R$k$NN of $q$ in $P$ can be defined as

$$
    RkNN(q) = \{p \in P \mid q \in kNN(p)\}.
$$

(3)

Since the vast majority of spatial data in the GIS applied in real life are two-dimensional and most of the applications of R$k$NN queries are in location-based services, like the existing approaches [2], [4], [5], [6], our study focus on the R$k$NN query in the 2D scene. In the following sections, if not specified, the problem is discussed in two-dimensional environment by default.

B. Voronoi diagram & Delaunay graph

Voronoi diagram [16], proposed by Rene Descartes in 1644, is a spatial partition structure widely applied in many science domains, especially spatial database and computational geometry. In a Voronoi diagram of $n$ points, the space is divided into $n$ regions corresponding to these points, which are called Voronoi cells. For each of these $n$ points, the corresponding Voronoi cell consists of all locations closer to that point than to any other. In other words, each point is the nearest neighbor of all the locations in its corresponding Voronoi cell. Formally, the above description can be stated as follows.

Fig. 1. a) Voronoi Diagram. b) Delaunay Graph
**Definition 4. Voronoi cell & Voronoi diagram:** Given a set $P$ of $n$ points, the Voronoi cell of a point $p \in P$, denoted as $V(P, p)$ or $V(p)$ for short, is defined as Equation (4):

$$V(P, p) = \{ q \mid \forall p' \in P \setminus \{ p \} : \text{dist}(p, q) \leq \text{dist}(p', q) \}$$

and the Voronoi diagram of $P$, denoted as $VD(P)$, is defined as Equation (5):

$$VD(P) = \{ V(P, p) \mid p \in P \}$$

The Voronoi diagram of a certain set $P$ of points, $VD(P)$, is unique.

**Definition 5. Voronoi neighbor:** Given the Voronoi diagram of $P$, for a point $p$, its Voronoi neighbors are the points in $P$ whose Voronoi cells share an edge with $V(P, q)$. It is denoted as $VN(P, q)$ or $VN(q)$ for short. Note that the nearest point in $P$ to $p$ is among $VN(q)$.

**Lemma 1.** Let $p_k$ be the $k$-th nearest neighbor of $q$, then $p_k$ is a Voronoi neighbor of at least one point of the $k-1$ nearest neighbors of $q$ (where $k > 1$).

**Proof.** See [7].

**Lemma 2.** For a Voronoi diagram, the expected number of Voronoi neighbors of a generator point does not exceed 6.

**Proof.** Let $n$, $n_e$ and $n_v$ be the number of generator points, Voronoi edges and Voronoi vertices of a Voronoi diagram in $\mathbb{R}^2$, respectively, and assume $n \geq 3$. According to Euler’s formula,

$$n + n_v - n_e = 1$$

Every Voronoi vertex has at least 3 Voronoi edges and each Voronoi edge belongs to two Voronoi vertices. Hence the number of Voronoi edges is not less than $3(n_v + 1)/2$, i.e.,

$$n_e \geq \frac{3}{2}(n_v + 1)$$

According to Equation (6) and Equation (7), the following relationships holds:

$$n_e \leq 3n - 6$$

When the number of generator points is large enough, the average number of Voronoi edges per Voronoi cell of a Voronoi diagram in $\mathbb{R}^d$ is a constant value depending only on $d$. When $d = 2$, every Voronoi edge is shared by two Voronoi Cells. Hence the average number of Voronoi edges per Voronoi cell does not exceed 6, i.e., $2 \cdot n_e/n \leq 2(3n - 6)/n = 6 - 12/n \leq 6$.

For set of points, a dual graph of its Voronoi Diagram is the Delaunay graph (graph of Delaunay triangulation) [17] of it.

**Definition 6. Delaunay triangulation & Delaunay graph:** For a set $P$ of discrete points in a plane, the Delaunay triangulation $DT(P)$ is that no point in $P$ is inside the circumcircle of any triangle of $DT(P)$. The graph of $DT(P)$ is called the Delaunay graph of $P$ and denoted as $DG(P)$.

Note that a graph of Delaunay graph must be a connected graph, i.e., two vertices in the graph are connected. For a set of points, its nearest neighbor graph is a subgraph of its Delaunay graph.

**Definition 7. Delaunay graph distance:** Given the Delaunay graph $DG(P)$, the Delaunay graph distance between two vertices $p$ and $p'$ of $DG(P)$ is the minimum number of edges connecting $p$ and $p'$ in $DG(P)$. It is denoted as $\text{dist}_{DG}(p, p')$.

**Lemma 3.** Given the query point $q$, if a point $p$ belongs to $R_k\text{NN}(q)$, then we have $\text{dist}_{DG}(p, q) \leq k$ in Delaunay graph $DG(p)$.

**Proof.** See [7].

**C. Conic section**

In mathematics, a conic section (or just conic) is a curve obtained as the intersection of a right circular conical surface with a plane. Conic curves include ellipse, hyperbola and parabola. Some properties of ellipses and hyperbolas are used in our work, so we introduce them in the follows.

**Definition 8. Ellipse:** An ellipse is a closed curve on a plane, such that the sum of the distances from any point on the curve to two fixed points $p_1$ and $p_2$ is a constant $C$. Formally, it is denoted as $E_{c:p_1:p_2}$ defined as follows:

$$E_{c:p_1:p_2} = \{ p \mid \text{dist}(p, p_1) + \text{dist}(p, p_2) = C \}$$

**Definition 9. Hyperbola:** A hyperbola is a geometric figure such that the difference between the distances from any point on the curve to two fixed points $p_1$ and $p_2$ is a constant $C$. Formally, it is denoted as $H_{c:p_1:p_2}$ defined as follows:

$$H_{c:p_1:p_2} = \{ p \mid |\text{dist}(p, p_1) - \text{dist}(p, p_2)| = C \}$$

**IV. METHODOLOGIES**

**A. Discriminance with Conic Section (CSD)**

**Definition 10. kNN region:** Given a query point $q$, the $k$NN region of $q$ is the inner region of $C_{q: \text{dist}(q, p_k)}$, i.e., the circle with $q$ as center and $\text{dist}(q, p_k)$ as the length of radius, where $p_k$ represents the $k$th closest point to $q$. This region is denoted as $RG_{k\text{NN}}(q)$. The radius of $RG_{k\text{NN}}(q)$ is called the $k$NN radius of $q$ and is denoted as $r_q$. 

![Fig. 2. kNN region](image-url)
Note that a point \( p \) must be one \( k\text{NN}(q) \) if it lies in \( RG_{\text{NN}}(q) \), i.e., the \( k\text{NN} \) region of \( q \). Conversely, if a point \( p' \) lies out of \( RG_{\text{NN}}(q) \), it cannot be any one of \( k\text{NN}(q) \). In Figure 2, \( q \) is the query point and the gray region within the circle centered on \( q \) represents \( RG_{\text{NN}}(q) \). As we can see, \( p_1 \), \( p_2 \) and \( p_3 \) lie inside \( RG_{\text{NN}}(q) \), then we can determine that they belong to \( k\text{NN}(q) \), while \( p_4 \) and \( p_5 \) lie outside. So they are not the members of \( k\text{NN}(q) \).

**Lemma 4.** Given a query point \( q \), a point \( p \) must be one of \( R_k\text{NN}(q) \) if it satisfies

\[
\text{dist}(p,q) \leq r_p.
\]  

(11)

Conversely, a point \( p' \) cannot be any one of \( R_k\text{NN}(q) \) if it satisfies

\[
\text{dist}(p',q) > r_{p'}.
\]  

(12)

Simply, for a point \( p \), if the query point \( q \) lies in its \( k\text{NN} \) region, \( p \) must be one of \( R_k\text{NN}(q) \), otherwise it must not belong to \( R_k\text{NN}(q) \).

**Proof.** The lemma is easily proved by the definition of \( k\text{NN} \) and \( R_k\text{NN} \), see Equation (2) and Equation (3). \[ \square \]

According to Lemma 4, we can determine whether a point \( p \) belongs to the \( R_k\text{NN} \) of the query point \( q \) by calculating the \( k\text{NN} \) region of \( p \). Obviously, \( q \) lying in \( RG_{\text{NN}}(p) \) is a necessary and sufficient condition for \( p \) to be one of \( R_k\text{NN}(q) \). In the refining phase of some \( R_k\text{NN} \) algorithms, the candidates are verified by this discriminant condition. In this discriminant, \( k\text{NN} \) region is required, so a \( k\text{NN} \) query must be conducted. The computational complexity of the state-of-the-art \( k\text{NN} \) algorithm is \( O(k \cdot \log k) \). Thus, the computational complexity of discriminant based on Lemma 4 is \( O(k \cdot \log k) \).

For most \( R_k\text{NN} \) algorithms, the size of candidate set is often several times much as that of the result set. Therefore, issuing a \( R_k\text{NN} \) verification of which the computational complexity is \( O(k \cdot \log k) \) for each candidate is obviously expensive. In order to reduce the computational cost of the refining phase of \( R_k\text{NN} \) queries, we introduce several more efficient verification approaches in the following.

**Lemma 5.** Given a query point \( q \) and a point \( p^+ \in R_k\text{NN}(q) \), a point \( p \) must be one of \( R_k\text{NN}(q) \) if it satisfies

\[
\text{dist}(p,q) + \text{dist}(p,p^+) \leq r_{p^+}.
\]  

(13)

**Proof.** As shown in Figure 3, the larger circle takes \( p^+ \) as the center and \( r_{p^+} \) as the radius, which represents the \( k\text{NN} \) region of \( p^+ \). \( L_{p^+,p^+} \) is a line segment passing through the point \( p \) with a length of \( r_{p^+} \). The smaller circle takes \( p \) as the center and \( \text{dist}(p,p^+) \) as the radius. Let \( p' \) be an arbitrary point inside \( C_{p^+\text{dist}(p,p^+)} \), then it must satisfy that

\[
\text{dist}(p,p') \leq \text{dist}(p,p^+).
\]  

(14)

According to the triangle inequality, we can obtain

\[
\text{dist}(p',p^+) \leq \text{dist}(p,p') + \text{dist}(p,p^+).
\]  

(15)

Combining Inequality (14) and Inequality (15), we can obtain

\[
\text{dist}(p',p^+) \leq \text{dist}(p,p^+) + \text{dist}(p,p^+)
\]

\[
= \text{dist}(p^+,p^+)
\]

\[
= r_{p^+}.
\]  

(16)

From above, we can construct a corollary that any point lying in \( C_{p^+\text{dist}(p,p^+)} \) must belong to \( k\text{NN}(p^+) \). Specifically, the number of points lying in \( C_{p^+\text{dist}(p,p^+)} \) must not be greater than \( k \), i.e., the size of \( k\text{NN}(p^+) \). Equivalently, there is no more than \( k \) points closer to \( p \) than \( p^+ \). Thus, \( p_k \) (the \( k \)th closest point to \( p \)) cannot be closer than \( p^+ \) to \( p \). Then \( \text{dist}(p,p^+) \leq \text{dist}(p,p_k) = r_p \). Suppose Inequality (13) holds,

\[
\text{dist}(p,q) \leq r_{p^+} - \text{dist}(p,p^+)
\]

\[
= \text{dist}(p^+,p^+) - \text{dist}(p,p^+)
\]

\[
= \text{dist}(p,p^+) \leq r_p.
\]  

(17)

From Lemma 4 and Inequality (17), we can deduce that \( p \in R_k\text{NN}(q) \). Therefore Lemma 5 is proved to be true. \[ \square \]

Lemma 5 provides a sufficient but unnecessary condition for determining that a point belongs to \( R_k\text{NN}(q) \), where \( q \) represents the query point. That means if a point \( p \) satisfies the condition of Inequality (13), it can be determined as one of \( R_k\text{NN}(q) \) without issuing a \( k\text{NN} \) query. In the case that \( r_{p^+} \) is known, we can verify whether Inequality (13) holds by only calculating the Euclidean distance from \( p \) to \( q \) and \( p^+ \) respectively. Calculating the Euclidean distance between two points can be regarded as an atomic operation. Hence the computational complexity of the discriminant corresponding to Lemma 5 is \( O(1) \).

**Definition 11.** Positive discriminant region: Given the query point \( q \) and a point \( p \), the positive discriminant region of \( p \) is the internal region of \( E_{p,q}^+ \). Formally, it is denoted as \( RG_{\text{disc}}^+(p) \) and is defined as follows:

\[
RG_{\text{disc}}^+(p) = \{ p' \mid \text{dist}(p',q) + \text{dist}(p',p) \leq r_p \}.
\]  

(18)

From the triangle inequality, it can be shown that

\[
\text{dist}(p',q) + \text{dist}(p',p) \geq \text{dist}(p,q).
\]  

(19)
Lemma 6. Given a query point \( q \) and a point \( p^- \notin \text{RkNN}(q) \), a point \( p \) cannot be any one of \( \text{RkNN}(q) \) if it satisfies

\[
dist(p, q) - dist(p, p^-) > r_{p^-}.
\]  

According to the triangle inequality, we can obtain

\[
dist(p, p') \leq dist(p, p^-) + dist(p^-, p').
\]  

From Inequality \((22)\) and Inequality \((23)\), we can get that

\[
dist(p, p') \leq dist(p, p^-) + dist(p^-, p^x) = dist(p, p^x) = r_p.
\]  

Then we realize that all the points lying in \( \text{RG}_k\text{NN}(p^-) \) must lie inside \( C_{p, \text{dist}(p, p^x)} \), namely the number of points lying inside of \( C_{p, \text{dist}(p, p^x)} \) must be no less than \( k \), i.e., the number of points lying in \( \text{RG}_k\text{NN}(p^-) \). That is to say, there exist at least \( k \) points no further than \( p^x \) away from \( p \). Equivalently,

\[
dist(p, p^x) \geq dist(p, p_k) = r_p \quad \text{(where \( p_k \) represents the \( k \)-th closest point to \( p \)).}
\]

If the condition of Inequality \((21)\) is satisfied,

\[
dist(p, q) > dist(p, p^-) + r_{p^-} = dist(p, p^-) + dist(p^-, p^x) = dist(p, p^x) \geq r_p.
\]

From Lemma \(5\) and Inequality \((25)\), we can deduce that \( p \notin \text{RkNN}(q) \). Therefore, Lemma \(6\) proved to be true.

From Lemma \(6\), we can know that, if a point is determined not to be one of \( \text{RkNN}(q) \) and its \( k \)-NN radius is known, then there may exist some other points that can be sufficiently determined to belong to \( \text{RkNN}(q) \) without performing a \( k \)-NN query but by performing two times of simple Euclidean distance calculation. That means the computational complexity of the discrimination based on Lemma \(6\) is \( O(1) \).

\[\]
If \( p \in \mathbb{R}^k \text{NN}(q) \), i.e., \( \text{dist}(p, q) \leq r_p \),
\[
\text{dist}(p', q) - \text{dist}(p', p) \leq \text{dist}(p, q) \leq r_p
\] (28)
then \( \mathcal{RG}_{\text{disc}}^+(p) = \emptyset \). Therefore, if \( \mathcal{RG}_{\text{disc}}^-(p) \) is not empty, \( p \) must belong to \( \mathbb{R}^k \text{NN}(q) \). Hence from Lemma 6, we can draw such a corollary that, for an arbitrary point \( p \), if \( \mathcal{RG}_{\text{disc}}^-(p) \) is not empty, any point lying inside \( \mathcal{RG}_{\text{disc}}^-(p) \) cannot belong to \( \mathbb{R}^k \text{NN}(q) \).

As shown in Figure 3, \( q \) represents the query point, the region within the circle centered on \( p \) represents \( \mathbb{R}^k \text{NN}(p) \), and the gray region separated by the hyperbola \( H_{p,q}^r \) on the right represents \( \mathcal{RG}_{\text{disc}}^-(p) \). As in the figure, \( p_1 \) and \( p_2 \) lie inside \( \mathcal{RG}_{\text{disc}}^-(p) \), while \( p_3 \) and \( p_4 \) do not. Then we can determine that \( p_1 \) and \( p_2 \) must not belong to \( \mathbb{R}^k \text{NN}(q) \), whereas we cannot tell by Lemma 6 whether \( p_3 \) or \( p_4 \) belongs to \( \mathbb{R}^k \text{NN}(q) \) or not.

**Definition 13. Positive/Negative discriminant point:** Given the query point \( q \) and two other points \( p \) and \( p' \), if \( p' \) lies in \( \mathcal{RG}_{\text{disc}}^+(p) \), we claim that \( p \) is a positive discriminant point of \( p' \) and \( p \) can positive discriminate \( p' \). It is denoted as \( p \xrightarrow{+ \text{disc}} p' \). Similarity, if \( p' \) lies in \( \mathcal{RG}_{\text{disc}}^-(p) \), we name that \( p \) is a negative discriminant point of \( p' \) and \( p \) can negative discriminate \( p' \). It is denoted as \( p \xrightarrow{- \text{disc}} p' \). If not specified, both of these two types of points may be collectively referred to as discriminant points and we can use \( p \xrightarrow{\text{disc}} p' \) to express that \( p \) can discriminate \( p' \).

Whether a point belongs to the \( \mathbb{R}^k \text{NN} \) set of the query point or not, the corresponding discriminant method with low computational complexity is provided. However, when performing the discrimination of Lemma 5 or Lemma 6, the distance from the point to be determined to the query point and the positive/negative discriminant point should be calculated respectively. In order to further improve the verification efficiency of some points, we propose Lemma 7.

**Lemma 7.** Given a query point \( q \), a point \( p \) must be one of \( \mathbb{R}^k \text{NN}(q) \) if it satisfies
\[
\text{dist}(p, q) \leq r_q/2.
\]

![Fig. 7. Lemma 7](image)

**Proof.** In Figure 7, there are three circles, two of which are centered on \( q \) and take \( r_q \) and \( r_q/2 \) as the length of their radii, respectively. The other circle takes \( p \) as the center and \( \text{dist}(p, q) \) as the length of the radius, where \( p \) lies in \( c_{q,r_q/2} \), i.e., \( \text{dist}(q, p) \leq r_q/2 \). Let \( p' \) be an arbitrary point inside of \( C_{p, \text{dist}(p, q)} \), then it must satisfy that
\[
\text{dist}(p, p') \leq \text{dist}(q, p).
\]
(29)
From the triangle inequality of \( \triangle qpp' \), it can be obtained that
\[
\text{dist}(q, p') \leq \text{dist}(q, p) + \text{dist}(p, p').
\]
(30)
Then we can get that,
\[
\text{dist}(q, p') \leq 2 \cdot \text{dist}(q, p).
\]
(31)
Because \( \text{dist}(q, p) \leq r_q/2 \),
\[
\text{dist}(q, p') \leq 2 \cdot r_q/2 = r_q
\]
(32)
That means, any point lying in \( C_{p, \text{dist}(p, q)} \) must belong to \( \mathbb{R}^k \text{NN}(q) \). Therefore, the number of points lying in \( C_{p, \text{dist}(p, q)} \) must not be greater than \( k \), i.e., the size of \( \mathbb{R}^k \text{NN}(q) \), which means there is no more than \( k \) points closer to \( p \) than \( q \). Hence \( p_k \) (\( k \)th closest point to \( p \)) cannot be closer than \( q \) to \( p \). Then
\[
\text{dist}(q, p') \leq \text{dist}(p, p_k) = r_p.
\]
(33)
According to Lemma 4, \( p \in \mathbb{R}^k \text{NN}(q) \), then Lemma 7 is proved.

![Fig. 8. Semi-k-NN region](image)

**Definition 14. Semi-k-NN region:** Given the query point \( q \), the semi-\( k \)-NN region of \( q \) is the internal region of \( C_{q,r_q/2} \). Formally, it is denoted as \( \mathcal{SRG}_{\text{kkNN}}(q) \) and is defined as Equation (34).
\[
\mathcal{SRG}_{\text{kkNN}}(q) = \{ p \mid \text{dist}(p, q) \leq r_q/2 \}
\]
(34)
As shown in Figure 8, \( q \) represents the query point, the region within the larger circle represents \( \mathcal{RG}_{\text{NN}}(q) \), and the gray region within the smaller circle represents \( \mathcal{SRG}_{\text{NN}}(q) \). It can be observed from the figure, \( p_1 \) and \( p_2 \) lie in the gray region, while \( p_3 \) and \( p_4 \) do not. Then \( p_1 \) and \( p_2 \) can be determined as members of \( \mathbb{R}^k \text{NN}(q) \). Nevertheless, we cannot determine whether \( p_3 \), \( p_4 \) or \( p_5 \) belongs to \( \mathbb{R}^k \text{NN}(q) \) or not by Lemma 7.

With Lemma 4, Lemma 5, Lemma 6 and Lemma 7, we can find all the points in the \( \mathbb{R}^k \text{NNs} \) of the query point by verifying only a small portion of points in the candidates. We combine these four lemmas to form a complete \( \mathbb{R}^k \text{NN} \) verification method named CSD (Conic Section Discriminant).
B. Selection of discriminant points

Theoretically, when using CSD to verify the candidates of a R\(k\)NN query, any point that belongs to the R\(k\)NN set can be considered as a positive discriminant point. Similarly, if a point is not a member of R\(k\)NNs, then it can be considered as a negative discriminant point. In other words, all points in the candidate set are eligible to be selected as discriminant points. Our aim is to issue as few \(k\)NN queries as possible in the process of R\(k\)NN queries, that is, to use as few discriminant points as possible to discriminate all the other points in the candidate set. Therefore, the selection of discriminant points is very important for improving the efficiency of R\(k\)NN queries. Which points should be selected as discriminant points is what we will scrutinize next.

**Definition 15. Discriminant set:** For a R\(k\)NN query, given a set \(S_{\text{end}}\) of candidates and denoted as \(S_{\text{dist}}\), a discriminant set is such a set that the following condition is satisfied:

\[
\forall p \in S_{\text{end}} \setminus S_{\text{dist}}, \exists p' \in S_{\text{dist}} : p' \xrightarrow{\text{disc}} p. \tag{35}
\]

Because it is not certain how many points and which points need to be selected as discriminant points, the total number of schemes for selecting discriminant points can be as large as \(\sum_{i=1}^{\left|S_{\text{end}}\right|} \left(S_{\text{end}} \setminus S_{\text{dist}}\right)\), where \(\left|S_{\text{end}}\right|\) means the number of candidates. Hence the computational complexity of finding the absolute optimal one out of all the schemes is as much as \(O(k!)\). However, it is not difficult to come up with a relatively good discriminant points selecting scheme, of which the size of the discriminant set \(\left|S_{\text{dist}}\right|\) is just about \(O(\sqrt{k})\).

For a positive discriminant point, most of the points in its discriminant region are closer to the query point than itself. Furthermore, any negative discriminant point is closer to the query point than most of the points in its own discriminant region. Therefore, a point belonging to R\(k\)NNs can rarely be discriminated by a point closer to the query point than itself, and the probability that a point not belonging to R\(k\)NNs can be discriminated by a point further than itself away from the query point is also very low. Therefore, the points which are extremely close to the boundary of the R\(k\)NN region (i.e., influence zone [5]) are rarely able to be discriminated by other points. Thus, these points should be selected as discriminant points in preference. However, it is impossible to directly find these points near the boundary without pre-calculating the R\(k\)NN region. Calculating the R\(k\)NN region is a very computational costly process for its computational complexity of \(O(k^3)\). While the \(k\)NN region of the query point is easy to be obtained by issuing a \(k\)NN computation. Assuming that the points are uniformly distributed, the \(k\)NN region and the R\(k\)NN region of a query point are extremely approximate and the difference between them is negligible. Hence it is a good strategy to preferentially select the points near the boundary of \(k\)NN region as the discriminant points to some extent.

As shown in Figure 9, there are some points distributed. The region inside the circle with \(q\) as the center represents the \(k\)NN region of \(q\). In general, only the points near the boundary of \(RG_{\text{kNN}}(q)\) need to be selected as the discriminant points and all the other candidate points can be discriminated by these discriminant points. In other words, if the points are evenly distributed, the points near the boundary of \(RG_{\text{kNN}}(q)\) are enough to form a valid discriminant set of \(q\). Because the distribution of points is not guaranteed to be absolute uniform, it is not always reliable if only the points near the boundary of the \(k\)NN region of the query point are taken as discriminant points for a R\(k\)NN query.

In order to ensure the reliability of the selection, we propose a strategy to dynamically construct the discriminant set while verifying the candidate points. First, the candidate points belonging to \(k\)NN(\(q\)) are accessed in descending order of distance to \(q\). Then the other candidate points are accessed in ascending order of distance to \(q\). During the process of accessing candidates, once the currently accessed point cannot be discriminated by any point in the discriminant set, this point should be selected as a discriminant point and put into the discriminant set. Otherwise, we can use a corresponding point in the discriminant set to determine whether it belongs to R\(k\)NNs or not.

C. Matching candidate points with discriminant points

Under the above strategy, it is sufficient to ensure that any point not belonging to \(S_{\text{dist}}\) can be discriminated by at least one point in \(S_{\text{dist}}\). Since the expected size of \(S_{\text{dist}}\) is \(O(\sqrt{k})\) (see Section 5), the computational complexity of finding a discriminant point for a point by exhaustive searching the discriminant set is \(O(\sqrt{k})\). Obviously, it is not a good idea to match candidate points with their discriminant point in this way. Therefore, we propose a method based on Voronoi diagrams to improve the efficiency of this process.

Given a Voronoi diagram \(VD(P)\) of a point set \(P\) and a continuous region \(RG\), the vast majority of points in \(RG\) have at least one Voronoi neighbor lying in \(RG\) [18]. For any discriminant point, its discriminant region is a continuous region (ellipse region or hyperbola region). So for a non-discriminant point, there is high probability that at least one of its Voronoi neighbors can discriminate it or shares a discriminant point with it. Therefore, when accessing a candidate point, if the point can be discriminated by one of its Voronoi neighbors
or the discriminant point of one of its Voronoi neighbors, this point can be determined whether belongs to the R$k$NNs. Otherwise, we say that this point is almost impossible to be discriminated by any known discriminant point and it should be marked as a discriminate point. Recall Lemma 2 in two dimensions, the expected number of Voronoi neighbors per point is 6, which is a constant. By using the above approach we can find the discriminant point for a non-discriminant point with a computational complexity of $O(1)$.

D. Algorithm

In the previous three subsections, we introduce a discriminator with conic sections (CSD) for improving the efficiency of R$k$NN queries, and its principle is also explained. In this subsection, we will introduce the implementation of the R$k$NN algorithm based on CSD.

The pseudocode for CSD is shown in Algorithm 1. When verifying a point, we first try to determine whether the point belongs to R$k$NNs by Lemma 4 (line 2). If this fails, we visit the Voronoi neighbors of the point and try to use Lemma 2 or Lemma 3 to discriminate it (line 10 and line 13). If none of the three lemmas above apply to this point, then we issue a NN query for it and use Lemma 4 to verify it (line 18).

Using CSD, we implement an efficient R$k$NN algorithm, as shown in Algorithm 2. First we generate the candidate set in the same way as VR-R$k$NN [7], where the size of candidate is $6k$ (line 1). Next, the candidate set is sorted in ascending order by the distance to the query point (line 2). Then the first $k$ elements of the candidate set and the rest of the elements are divided into two groups. The elements in the two groups are verified one by one in the order from back to front and from front to back, respectively (line 8 and line 11). After all candidate points are verified, the R$k$NNs of the query point is obtained.

We used the same algorithm as VR-R$k$NN to generate the candidate set, and we do not improve it. The core of this algorithm is still from the Six-regions [2]. In addition, it uses a Voronoi diagram to find the candidate points incrementally according to Lemma 1. By Lemma 3, only the points whose Delaunay distance to the query point is not larger than $k$ are eligible to be selected as candidate points. Hence the number of points accessed for finding candidates in the algorithm is guaranteed to be no more than $O(k^2)$. The pseudocode of the algorithm for generating candidates is presented in Algorithm 3. As generating the candidate set is not the focus of our study, we do not describe Algorithm 3 in detail. [7] can be referred to for specific instructions.

V. THEORETICAL ANALYSIS

In this section, we analyze the expected size of discriminant set, the expected number of accessed points and the computational complexity of CSD-R$k$NN. We assume that the data points are uniformly distributed in a unit space.

A. Expected size of discriminant set

The query point is $q$, the number of points in R$k$NN($q$) is $|R$k$NN|$, and the number of points near the boundary of

**Algorithm 1: CSD($p, q, k, r_q, S_v, S_{disc}, D_{disc}$)**

**Input:** the point $p$ to be verified, the query point $q$, the parameter $k$, the $k$NN radius $r_q$ of $q$, the set $S_v$ of points that have been visited, the discriminant set $S_{disc}$ and the dictionary $D_{disc}$ that records the corresponding discriminant points for non-discriminant points

**Output:** whether $p \in R$k$NN($q$).

```plaintext
1 $S_v$.add($p$);
2 if dist($p, q$) ≤ $r_q$/2 then /* Lemma 7 */
3 return true;
4 foreach $p_n \in VN(p)$ do
5 if $p_n \in S_v$ then
6 if $p_n \in S_{disc}$ then
7 $p_{disc} \leftarrow p_n$;
8 else
9 $p_{disc} \leftarrow D_{disc}[p_n]$;
10 if $p_{disc} \in R$k$NN($q$) and dist($p, q$) +
11 dist($p, p_{disc}$) ≤ $r_{disc}$ then /* Lemma 5 */
12 $D_{disc}[p] \leftarrow p_{disc}$;
13 return true;
14 if $p_{disc} \notin R$k$NN($q$) and dist($p, q$) −
15 dist($p, p_{disc}$) > $r_{disc}$ then /* Lemma 6 */
16 $D_{disc}[p] \leftarrow p_{disc}$;
17 return false;
```

**Algorithm 2: CSD-R$k$NN($q$)**

**Input:** the query point $q$

**Output:** R$k$NN($q$)

```plaintext
1 $S_{cnd} \leftarrow$ generateCandidates($q$, $k$);
2 Sort $S_{cnd}$ in ascending order by the distance to $q$;
3 $r_q \leftarrow$ calculate the $k$NN radius of $q$;
4 $S_v \leftarrow \emptyset$;
5 $S_{disc} \leftarrow \emptyset$;
6 $D_{disc} \leftarrow$ generate an empty dictionary;
7 $S_{R$k$NN} \leftarrow \emptyset$;
8 for $i \leftarrow k$ to 1 do
9 if CSD($S_{cnd}[i], q, k, r_q, S_v, S_{disc}, D_{disc}$) then
10 $S_{R$k$NN}.add(S_{cnd}[i])$;
11 for $i \leftarrow k + 1$ to $6k$ do
12 if CSD($S_{cnd}[i], q, k, r_q, S_v, S_{disc}, D_{disc}$) then
13 $S_{R$k$NN}.add(S_{cnd}[i])$;
14 return $S_{R$k$NN}$;
```
Algorithm 3: generateCandidates($q, k$)

Input: the query point $q$ and the parameter $k$

Output: the candidates of R$k$NN($q$)

1. $H \leftarrow \text{MinHeap}()$;
2. $\text{Visited} \leftarrow \emptyset$;
3. for $i \leftarrow 1$ to $6$ do
4.   $S_{\text{end}}[i] \leftarrow \text{MinHeap}()$;
5.   foreach $p \in VN(q)$ do
6.     $H.push([1, p])$;
7.     $\text{Visited.add}(p)$;
8.   while $|H| > 0$ do
9.     $[\text{dist}_{DG}(p), p] \leftarrow H.pop()$;
10. for $i \leftarrow 1$ to $6$ do
11.   if $\text{Segment}_i$ contains $p$ then
12.     if $|S_{\text{end}}[i]| > 0$ then
13.       $p_n \leftarrow$ the last point in $S_{\text{end}}[i]$;
14.     else
15.       $p_n \leftarrow$ a point infinitely away from $q$;
16.     if $\text{dist}_{DG}(p) \leq k$ and
17.        $\text{dist}(q, p) \leq \text{dist}(q, p_n)$ then
18.        $S_{\text{end}}[i].push([\text{dist}(p, q), p])$;
19.   foreach $p' \in VN(p)$ do
20.     if $p' \notin \text{Visited}$ then
21.       $\text{dist}_{DG}(p') \leftarrow \text{dist}_{DG}(p) + 1$;
22.       $H.push([\text{dist}_{DG}(p'), p'])$;
23.       $\text{Visited.add}(p')$;
24.   $\text{Candidates} \leftarrow \emptyset$;
25. for $i \leftarrow 1$ to $6$ do
26.   for $j \leftarrow 1$ to $k$ do
27.     $\text{Candidates.add}(S_{\text{end}}[i].pop())$;
28. return $\text{Candidates}$;

$R_{\text{kNN}}(q)$ is $|S_b|$. The area and circumference (total length of the boundary) of $R_{\text{kNN}}(q)$ are denoted as $A_{\text{kNN}}(q)$ and $C_{\text{kNN}}(q)$, respectively. The expected size of the discriminant set of $q$ is $|S_{\text{disc}}|$.

It is shown that the expected value of $|RkNN|$ is $k$ [5]. Thus, the radius of the approximate circle of $R_{\text{kNN}}(q)$ is equal to $r_q$. Then

$$A_{\text{kNN}}(q) = \pi \cdot r_q^2$$  \hspace{1cm} (36)

$$C_{\text{kNN}}(q) = 2\pi \cdot r_q$$  \hspace{1cm} (37)

The following equation can be obtained from Equation (36) and Equation (37).

$$C_{\text{kNN}}(q) = 2\sqrt{\pi} \cdot A_{\text{kNN}}(q)$$  \hspace{1cm} (38)

As the points around the boundary of $R_{\text{kNN}}(q)$ consists of two sets of points where one is inside $R_{\text{kNN}}(q)$ and the other is outside, $|S_b|$ is to $|RkNN|$ what $2 \cdot C_{\text{RkNN}}(q)$ is to $A_{\text{RkNN}}(q)$, i.e.,

$$|S_b| = 2 \cdot 2\sqrt{\pi} \cdot |RkNN| = 4\sqrt{\pi} \cdot k \approx 7.1\sqrt{k}.$$  \hspace{1cm} (39)

If all the points near the boundary are selected as the discriminant points, there must be some redundancy, i.e., the discriminant region of some points will overlap. Hence the size of the discriminant set generated under our strategy is less than the number of the points near the boundary of the R$k$NN region, i.e., $|S_{\text{disc}}| \leq 7.1\sqrt{k}$.

B. Expected number of accessed points

For an R$k$NN query of $q$, the candidate points are distributed in an approximately circular region $RG_{\text{end}}(q)$ centered around $q$, which has an area $A_{\text{end}}(q)$ and a circumference $C_{\text{end}}(q)$. The expected number of accessed points is $|S_{\text{ac}}|$. In the filtering phase of CSD-R$k$NN, the points accessed include all the the candidate points and their Voronoi neighbors. Except for the points in the candidate set, the other accessed points are distributed outside $RG_{\text{end}}(q)$ and adjacent to the boundary of $RG_{\text{end}}(q)$. Hence $|S_{\text{ac}}| - |S_{\text{end}}| = |S_{\text{end}}|$ what $C_{\text{end}}(q)$ is to $A_{\text{end}}(q)$, i.e.,

$$|S_{\text{ac}}| - |S_{\text{end}}| = 2\sqrt{\pi} \cdot |S_{\text{end}}|$$  \hspace{1cm} (40)

$$|S_{\text{ac}}| = |S_{\text{end}}| + 2\sqrt{\pi} \cdot |S_{\text{end}}|$$

$$= 6k + 2\sqrt{\pi} \cdot 6k \approx 6k + 8.7\sqrt{k}$$  \hspace{1cm} (41)

Therefore, if the points are distributed uniformly, the expected number of accessed points is approximately $6k + 8.7\sqrt{k}$. When the points are distributed unevenly, $|S_{\text{ac}}|$ becomes larger. However, it has an upper bound. Recall Lemma [3] we can make deduce that only the points whose Delaunay graph distance to $q$ is not larger than $k$ are eligible to be selected as candidate points. Then

$$|S_{\text{ac}}| \leq \sum_{i=1}^{k} 2\pi \cdot i = (k^2 + k)\pi.$$  \hspace{1cm} (42)

C. Computational complexity

The expected computational complexity of the filtering phase of CSD-R$k$NN is $O(k \cdot \log k)$ [7]. In the refining phase, we have to issue a $k$NN query with $O(k \cdot \log k)$ computational complexity for each discriminant point, and the size of the discriminant set is about $7.1\sqrt{k}$. The other candidates only need to be verified by CSD. Thus, the computational complexity of the refining phase is $O(k^{1.5} \cdot \log k)$. Hence the overall computational complexity of CSD-R$k$NN is $O(k^{1.5} \cdot \log k)$.

VI. EXPERIMENTS

In the previous section, we discussed the theoretical performance of CSD-R$k$NN. In this section, we intend to evaluate the performance of aspects through comparison experiments.
A. Experimental settings

In the experiments, we let VR-RkNN [7] and the state-of-the-art RkNN approach SLICE [6] to be the competitors of our method.

The settings of our experiment environment are as follows. The experiment is conducted on a personal computer with Python 2.7. The CPU is Intel Core i5-4308U 2.80GHz and the RAM is DDR3 8G.

To be fair, all three methods in the experiment are implemented in Python, with six partitions in the pruning phase. We use two types of experimental data sets: simulated data set and real data set[1]. To decrease the error of the experiments, we repeat each experiment for 30 times and calculate the average of the results. The query point for each time of the experiment is randomly generated.

Our experiments are designed into four sets. The first set of experiments is used to evaluate the effect of the data size on the time cost of the RkNN algorithms. The data size is from 10^3 to 10^6 and the value of k is fixed at 200. The rest of sets are used to evaluate the effect of the value of k on the time cost, the number of verified points and the number of the accessed points of the RkNN algorithms, respectively. For these three sets of experiments, the size of the simulated data is fixed at 10^6, the size of the real data is 49,601 and the value of k varies from 10^1 to 10^4.

B. Experimental results

![Table II](image)

**Table II**

**Total time cost (in ms) of different RkNN algorithms with various sizes of data sets.**

| Algorithm     | Data size | 10^3 | 10^4 | 10^5 | 10^6 |
|---------------|-----------|------|------|------|------|
| VR-RkNN       |           | 510  | 725  | 728  | 732  |
| SLICE         |           | 232  | 397  | 438  | 441  |
| CSD-RkNN      |           | 59   | 65   | 69   | 72   |

![Figure 10](image)

**Figure 10.** Effect of data size on efficiency of RkNN queries

Figure [10] shows the time cost of the three RkNN algorithms with various data sizes. As we can see, when the number of points in the database is significantly much larger than k, the impact of the data size on the time cost of RkNN queries is very limited. If the number of points in the database is small enough to be on the same order of magnitude as k, all points in the database become candidate points. Then the smaller the database size, the less time cost of the RkNN query. When the number of points in the database is above 10,000 and the value of k is fixed at 200, the time cost of CSD-RkNN is always around 84% and 90% less than that of SLICE and VR-RkNN, respectively. The detailed experimental results are presented in Table [II].

![Figure 11](image)

**Figure 11.** Effect of k on efficiency of RkNN queries

Figure [11] shows the influence of k on the efficiency of these three RkNN algorithms, where sub-figure (a) and (b) shows the time cost of RkNN queries from simulated data and real data, respectively. As k varies from 10 to 10,000, the time cost of these three algorithms increases. With both synthetic data and real data, the query efficiency of CSD-RkNN is significantly higher than that of the other two competitors. With the increase of k, this advantage becomes more and more obvious. When k is 10,000, the time cost of CSD-RkNN is only about 1/10 of that of the state-of-the-art algorithm SLICE. The detailed experimental results are presented in Table [III].

![Table III](image)

**Table III**

**Total time cost (in ms) of RkNN queries with various values of k.**

| k     | Simulated data | Real data |
|-------|----------------|-----------|
|       | VR-RkNN | SLICE  | CSD-RkNN | VR-RkNN | SLICE  | CSD-RkNN |
| 10^1  | 5       | 26     | 2        | 4       | 28     | 2        |
| 10^2  | 199     | 193    | 39       | 194     | 283    | 29       |
| 10^3  | 20576   | 3759   | 801      | 17212   | 4610   | 813      |
| 10^4  | 2118391 | 321233 | 22077    | 1829742 | 226959 | 23911    |

![Table IV](image)

**Table IV**

**Number of candidates verified by RkNN algorithms with various values of k.**

| k     | Simulated data | Real data |
|-------|----------------|-----------|
|       | VR-RkNN | SLICE  | CSD-RkNN | VR-RkNN | SLICE  | CSD-RkNN |
| 10^1  | 60      | 25     | 20       | 60      | 20     | 17       |
| 10^2  | 600     | 257    | 57       | 600     | 203    | 46       |
| 10^3  | 6000    | 2572   | 186      | 6000    | 2257   | 156      |
| 10^4  | 60000   | 25675  | 599      | 49601   | 23874  | 627      |

149,601 non-duplicative data points on the geographic coordinates of the National Register of Historic Places [http://www.math.uwaterloo.ca/~usu/files/us50000_latlong.txt](http://www.math.uwaterloo.ca/~usu/files/us50000_latlong.txt)
Figure 12 reflects the relationship between $k$ and the number of candidate points verified by the three algorithms in the experiments. Sub-figure (a) and (b) show the experimental results on simulated data and real data, respectively. These two sub-figures also show the theoretical number of candidate points verified with different values of $k$. During the execution of CSD-R$k$NN, only the points in the discriminant set are verified by issuing $k$NN queries. Therefore, the number of candidates verified is equal to the size of the discriminant set. As we discussed in section V-A, the size of the discriminant set is theoretically not larger than $7.1\sqrt{k}$. In consequence, the theoretical number of verified candidates in Figure 12 is $7.1\sqrt{k}$. It can be seen from the figure that the actual number of points verified is slightly less than the theoretical value, $7.1\sqrt{k}$. It indicates that the experimental results are consistent with our analysis. It is also obvious from the figure that the number of verified candidate points of CSD-R$k$NN is much smaller than that of the other two algorithms. The detailed experimental results are presented in Table V.

![Figure 12](image)

Fig. 12. Effect of $k$ on the number of candidates verified

Figure 13 shows the number of accessed points of the three algorithms in the experiments and the theoretical number of accessed points of CSD-R$k$NN with various values of $k$, which indirectly reflects their IO cost. It can be seen from sub-figure (a), the number of accessed points of the three algorithms is almost equal in terms of magnitude, and so is the theoretical value of CSD-R$k$NN. Specifically, the number of accessed points of CSD-R$k$NN is slightly smaller than that of SLICE. As shown in sub-figure (b), CSD-R$k$NN needs to access more points than SLICE. The reason is that the distribution of real data is very uneven, and CSD-R$k$NN is more sensitive to the distribution of data than SLICE. Note that CSD-R$k$NN and VR-R$k$NN use the same candidate set generation method, so they have almost the same number of accessed points. The detailed experimental results are presented in Table V.

![Figure 13](image)

Fig. 13. Effect of $k$ on the number of points accessed

From the above three experiments, it can be seen that R$k$NN query efficiency is little affected by the data size, but greatly affected by the value of $k$. CSD-R$k$NN is significantly more efficient than other algorithms because it requires less verification of candidate points. For data sets with very uneven distribution of points, the candidate set of CSD-R$k$NN is relatively large, which will affect the IO cost to some extent. However, the main time cost of the R$k$NN query is caused by a large number of verification operations rather than IO. Therefore, the distribution of points has little impact on the overall performance of CSD-R$k$NN.

VII. CONCLUSIONS AND FUTURE WORKS

In this paper, we propose CSD, a discriminant method to determine points whether belong to the R$k$NN set without issuing any queries with non-constant computational complexity. An efficient R$k$NN algorithm, named CSD-R$k$NN, is also implemented by using CSD. The comparative experiments are conducted between CSD-R$k$NN and other two R$k$NN algorithms of the state-of-the-art. The experimental results show that CSD-R$k$NN significantly outperforms its competitors in various aspects, except that CSD-R$k$NN needs to access more points to generate the candidate set when the distribution of points is very uneven. However, CSD-R$k$NN does not require costly validation of each candidate point. Hence the distribution of data has very limited impact on its overall performance.

As an efficient discriminant method for improving R$k$NN queries, CSD has a great potential and can be further developed. Our plan in the future is to extend the application of CSD to other variants of R$k$NN, including multidimensional R$k$NN, continuous R$k$NN, constrained R$k$NN, etc.

ACKNOWLEDGMENTS

The work is supported by the National Natural Science Foundation of China (U1711267, 41572314, 41972306), Hubei Province Innovation Group Project (2019CFA023) and Fundamental Research Funds for the Central Universities, China University of Geosciences (Wuhan) (CUGCJ1810).
REFERENCES

[1] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA, pages 201–212, 2000.

[2] Ioana Stanoi, Divyakant Agrawal, and Amr El Abbadi. Reverse nearest neighbor queries for dynamic databases. In 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, Texas, USA, May 14, 2000, pages 44–53, 2000.

[3] Yufei Tao, Dimitris Papadias, and Xiang Lian. Reverse knn search in arbitrary dimensionality. In Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB 04, page 744755. VLDB Endowment, 2004.

[4] Wei Wu, Fei Yang, Chee Yong Chan, and Kian-Lee Tan. FINCH: evaluating reverse k-nearest-neighbor queries on location data. PVLDB, 1(1):1056–1067, 2008.

[5] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang. Influence zone: Efficiently processing reverse k nearest neighbors queries. In Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany, pages 577–588, 2011.

[6] Shiying Yang, Muhammad Aamir Cheema, Xuemin Lin, and Ying Zhang. SLICE: reviving regions-based pruning for reverse k nearest neighbors queries. In IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014, pages 760–771, 2014.

[7] Mehdi Sharifzadeh and Cyrus Shahabi. Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. Proc. VLDB Endow., 3(1-2):1231–1242, September 2010.

[8] Anil Maheshwari, Jan Vahrenhold, and Norbert Zeh. On reverse nearest neighbor queries. In Proceedings of the 14th Canadian Conference on Computational Geometry, University of Lethbridge, Alberta, Canada, August 12-14, 2002, pages 128–132, 2002.

[9] Congjun Yang and King-Ip Lin. An index structure for efficient reverse nearest neighbor queries. In Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany, pages 485–492, 2001.

[10] Congjun Yang and King-Ip Lin. An index structure for efficient reverse nearest neighbor queries. In Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany, pages 485–492, 2001.

[11] King-Ip Lin, Michael Nolen, and Congjun Yang. Applying bulk insertion techniques for dynamic reverse nearest neighbor problems. In 7th International Database Engineering and Applications Symposium (IDEAS 2003), 16-18 July 2003, Hong Kong, China, pages 290–297, 2003.

[12] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, SIGMOD’84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984, pages 47–57. ACM Press, 1984.

[13] Giorgi R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. ACM Trans. Database Syst., 24(2):265–318, 1999.

[14] Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. Aggregate nearest neighbor queries in spatial databases. ACM Trans. Database Syst., 30(2):529–576, 2005.

[15] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. ACM Trans. Database Syst., 30(1):41–82, 2005.

[16] Cyrus Shahabi and Mehdi Sharifzadeh. Voronoi diagrams for query processing. In Encyclopedia of GIS, pages 2446–2452. Springer, 2017.

[17] B. Delaunay. Sur la sphère vide. a la mémoire de georges voronoi. Bulletin de l'Académie des Sciences de l'URSS. Classe des Sciences Mathématiques et Naturelles, 6:793–800, 1934.

[18] Yang Li. Area queries based on voronoi diagrams. Corr, abs/1912.00426, 2019.