ABSTRACT
The reverse $k$ nearest neighbors (R$k$NN) query is a prominent yet time-consuming spatial query used in facility siting, influential domain analysis, potential customer analysis, etc. Its aim is to identify all points that consider the query point as one of their $k$ closest points. However, when $k$ is relatively large (e.g. $k = 1000$), existing R$k$NN techniques often struggle to provide acceptable response times (within a few seconds). To address this issue, we propose a verification approach called conic section discriminance (CSD). This method serves to determine whether points belong to the R$k$NN set. With CSD, only a small fraction of candidates require costly $k$ nearest neighbors (kNN) queries for verification, while the rest can be rapidly verified with $O(1)$ complexity. Furthermore, we propose a Voronoi-based candidate generation approach to curtail the candidate set size. By leveraging the VoR-tree structure, we integrate these two approaches to form a novel R$k$NN algorithm named CSD-R$k$NN. A comprehensive set of experiments is conducted to compare CSD-R$k$NN with SLICE as the state-of-the-art R$k$NN algorithm, and VR-R$k$NN as the original R$k$NN algorithm on VoR-tree. The results indicate that CSD-R$k$NN consistently outperforms the other two algorithms, especially when $k$ is relatively large.

ARTICLE HISTORY
Received 11 October 2022
Accepted 15 August 2023

KEYWORDS
Spatial query; facility location; reverse $k$ nearest neighbors; conic section discriminance; Voronoi diagram

1. Introduction
Tobler’s first law of geography reads ‘everything is related to everything else, but near things are more related than distant things (Tobler 1970, 2004)’. Therefore, when analyzing the essential characteristics of a spatial object or its relationship with other things, the objects located nearby usually need to be retrieved first. To satisfy this requirement, most geographic information systems (GIS) and location-based services (LBS) provide two types of spatial queries for nearest neighbor analysis: the $k$ nearest neighbors (kNN) query and the reverse $k$ nearest neighbors (R$k$NN) query. We denote...
a point set and a query point as \( P \) and \( q \), respectively. A \( k \)NN query \( k\text{NN}(P, q) \) aims to retrieve from \( P \) the top \( k \) points closest to \( q \) (Fix and Hodges 1989). For \( Rk\text{NN} \) queries, all points that consider the query point as one of their top \( k \) closest points are required to be found.

The \( k \)NN query is extensively utilized in spatial applications for spatial nearest neighbor analysis, owing to its practicality and ease of implementation. As shown in Figure 1(a), if a pedestrian wants to find a nearby place to have lunch, the \( k \)NN query can help him find the \( k \) closest restaurants as options. However, in specific scenarios like facility siting (Zaheer et al. 2022), influential domain analysis (Wang et al. 2018) and potential customer analysis (Azri et al. 2020), \( Rk\text{NN} \) queries may be more suitable. Figure 1(b) illustrates how an \( Rk\text{NN} \) query can assist a restaurant owner in identifying potential customers. By utilizing this query, the owner can find all pedestrians who consider his restaurant as one of the \( k \) closest restaurants. These individuals may be more inclined to patronize his restaurant due to its proximity. In reality, when selecting a restaurant, people may take into account various factors beyond distance, including food taste, ambience, cleanliness, service quality, business scale and opening hours. Hence, when using \( k\text{NN} \) queries to find a restaurant, it is advisable to set a relatively large value for \( k \). Otherwise, there is a possibility of not finding any restaurant among the \( k \) nearest options that align with one’s preferences. Similarly, when conducting potential customer analysis for a restaurant using \( Rk\text{NN} \) queries, it is generally recommended to use a relatively large value for \( k \) to obtain a more comprehensive view of potential customers.

Existing \( Rk\text{NN} \) search techniques can generally return correct results in a relatively short time (a few seconds) when faced with small \( k \)-value query tasks. Unfortunately, once the value of \( k \) becomes relatively large, e.g. \( k \geq 1000 \), they can no longer respond in a tolerable time. In view of this issue, the main goal of this paper is to improve the performance of \( Rk\text{NN} \) algorithms when confronted with large \( k \)-value query tasks.

Most of the mainstream \( Rk\text{NN} \) search techniques consist of two crucial phases: pruning and verification. In the classical framework of \( Rk\text{NN} \) algorithms, these two phases are typically treated as independent of each other. In the phase of pruning,
most non-RkNN points, i.e. points which do not belong to the RkNN set, should be eliminated. The purpose is to obtain the smallest possible candidate set. Most RkNN algorithms suffer from high computational costs because their candidates need verification conducted by kNN queries or range queries that require a large computational overhead. SLICE is one of the state-of-the-art RkNN algorithms. It stands out as it employs a more efficient verification method that reduces the complexity of verifying each candidate to $O(k)$. However, the size of the candidate set obviously cannot be smaller than that of the RkNN set, which has been shown to be $O(k)$ (Cheema et al. 2011). Hence, performing verification for every candidate using this method still demands a significant amount of time. We have observed that existing RkNN algorithms share a common feature: the verification of each candidate is conducted independently. Therefore, the overall efficiency of the verification phase could be improved to some extent if verified candidates were utilized to support the verification of other unverified candidates.

Suppose that there is an RkNN point $p'$ near another point $p$. Compared with $p'$, if $p$ is more ambient to a query point $q$, $p$ is likely to be an RkNN point of $q$. If $p'$ is a non-RkNN point near $p$, and the distance between $p$ and $q$ is larger than that between $p'$ and $q$, then $p$ is likely to be a non-RkNN point of $q$. However, this intuition is still arbitrary and requires theoretical proof. Therefore, along with this idea, we further study and use the properties of conic sections to come up with a set of efficient RkNN verification methods, termed conic section discriminant (CSD). In CSD, a verified reference point is used to decide if its ambient point is an RkNN point. Note that its computational complexity is only $O(1)$. Then, for an RkNN algorithm, only the candidates selected as reference points need to be completely verified by the traditional verification method, while each of the other candidates can be directly determined as an RkNN point or a non-RkNN point with the help of a corresponding reference point. Furthermore, according to the distribution characteristics of RkNN points on Voronoi diagrams, a novel approach for generating RkNN candidates based on VoR-tree (Sharifzadeh and Shahabi 2010) is proposed to reduce the number of candidates as much as possible. We integrate the proposed candidate generation method with CSD to form a novel RkNN algorithm and name it CSD-RkNN. Extensive experiments have been conducted to evaluate the performance of the proposed algorithm. Compared with SLICE and VR-RkNN (the original RkNN algorithm on VoR-tree), CSD-RkNN demonstrates better performance, especially for large $k$-value RkNN queries.

2. Related works

The reverse nearest neighbor (RNN) query was initially proposed by Korn and Muthukrishnan (2000). Subsequently, its generalized form, the RkNN query, was introduced and extensively studied. Early algorithms for RkNN queries primarily relied on preprocessing techniques (Korn and Muthukrishnan 2000, Maheshwari et al. 2002, Lin et al. 2003). In recent years, non-preprocessing algorithms for RkNN queries have emerged, incorporating pruning and verification processes. These algorithms can be categorized into two main types based on their pruning methods: region-based pruning and half-space pruning. The technique of region-based pruning originated from an
algorithm known as ‘six-region’ (Stanoi et al. 2000). As its name suggests, six-regions dissect a space into six equal sections with the angle between each next two boundary rays 60°. In each section, only the closest point to the query point is likely to be an RNN point. Thus, this algorithm conducts separate NN queries for each of the six sections to obtain six candidates, which are then individually validated to obtain the final RNN set. TPL (Tao et al. 2004) is the first RkNN technique to use the half-space pruning strategy. It utilizes the bisector between the query point \( q \) and an arbitrary point \( p \) to prune the space. This bisector dissects the space perpendicularly in two half-planes. The half-plane containing \( p \) is \( PL_p(p, q) \) and the one containing \( q \) is \( PL_q(p, q) \). For a point \( p' \) in \( PL_p(p, q) \), it is evident that the distance between it and \( p \) is closer to that between it and \( q \), so \( p' \) is not an RNN point of \( q \). That is to say, \( p \) prunes \( p' \). Furthermore, \( p' \) is not an RkNN point if it is pruned by \( k \) other points or even more. For a significant period of time after the appearance of TPL, the field of RkNN was dominated by algorithms based on half-space pruning, which gave rise to several notable advancements (Wu et al. 2008, Cheema et al. 2011). However, the emergence of SLICE changed this landscape by reinvigorating the use of region-based pruning (Yang et al. 2014). SLICE employs an advanced pruning technique, which prunes a significantly larger area compared to six-region. Moreover, it enhances the verification process by generating a list of significant points, known as \( sigList \), for every section. By accessing the corresponding \( sigList \), each candidate can be verified with a complexity of \( O(k) \).

Most of the previous RkNN algorithms were implemented based on the R-tree. In contrast, VR-RkNN is an RkNN algorithm based on the VoR-tree (Sharifzadeh and Shahabi 2010). It achieves better I/O performance by utilizing the Voronoi diagram of the VoR-tree.

Other than the standard RkNN queries, a variety of RkNN query variants have been extensively studied, such as continuous RkNN (Cheema et al. 2012), visible RkNN (Gao et al. 2009), obstructed RkNN (Gao et al. 2016), privacy-preserving RkNN (Pournajaf et al. 2018), to name just a few. Moreover, RkNN query is widely used in many fields, such as facility siting (Chen et al. 2016, Lin et al. 2016, Zaheer et al. 2022), influential domain analysis (Wang et al. 2018), potential customer analysis (Azri et al. 2020), etc.

3. Methodology

This section consists of three subsections. The first subsection formalizes the core research problem, i.e. RkNN queries. The subsequent two subsections describe the proposed verification and pruning approaches for RkNN queries, respectively.

3.1. Problem settings

In practice, there are two types of RkNN queries: bichromatic RkNN (Bi-RkNN) and monochromatic RkNN (Mono-RkNN). Their formal definitions are as follows:

**Definition 3.1 (Bi-RkNN Queries).** Given a facility set \( F \), a user set \( U \), and a query facility \( q \in F \), a Bi-RkNN query retrieves each user that has \( q \) as one of its \( k \) nearest facilities from \( U \). Mathematically, it is defined as

\[
\text{Bi-RkNN}(U, F, q) = \{ u \in U | q \in k\text{NN}(F, u) \}. \tag{1}
\]
Definition 3.2 (Mono-RkNN Queries). Given a facility set $F$ and a query facility $q \in F$, a Mono-RkNN query retrieves each facility that has $q$ as one of its $k$ nearest facilities from $F$. Mathematically, it is defined as

$$
\text{Mono-RkNN}(F, q) = \{f \in F | q \in \text{KNN}(F, f)\}.
$$

Apparently, the Mono-RkNN query can be considered as a special case of the Bi-RkNN query when $U = F$. Hence as follows, we focus on the implementation of Bi-RkNN queries and briefly introduce how to answer Mono-RkNN queries. Unless otherwise specified, all discussions of RkNN are limited to the bichromatic mode in Euclidean space. Throughout the subsequent content, numerous notations are employed. To enhance reader comprehension, a summary of these notations and their respective meanings is provided in Table 1.

3.2. Conic section discriminance

First and foremost, we introduce a significant concept known as conic sections. They are curves that result from the intersection of a double-napped cone and a plane. Conic sections encompass four distinct types, namely circles, ellipses, hyperbolas and parabolas. The verification method proposed in this paper is closely related to the first three types, so we will provide detailed mathematical definitions for each of them.

Definition 3.3 (Circle). As shown in Figure 2(a), a circle (or sphere in multi-dimensional space) $C_o^q$ is a geometric shape such that the distance from any point on it to a fixed point (the center) $o$ is a constant (the radius) $r$. Formally,

$$
C_o^q = \{p | d(p, o) = r\}.
$$

Definition 3.4 (Ellipse). As shown in Figure 2(b), an ellipse (or an ellipsoid in a multi-dimensional space) $E_{f_1,f_2}^l$ is a geometric shape such that the sum of the distances from

| Notation | Description |
|----------|-------------|
| $d(p_1, p_2)$ | The Euclidean distance between points $p_1$ and $p_2$ |
| $C_o^r$ | A circle with $o$ as its center and $r$ as its radius |
| $E_{f_1,f_2}^l$ | An ellipse with foci at $f_1$ and $f_2$ and a major axis of length $l$ |
| $H_{f_1,f_2}(u)$ | A hyperbola with foci at $f_1$ and $f_2$ and a major axis of length $l$ |
| $\text{KNN}(F, u)$ | The KNN region of user $u$ w.r.t. the facility set $F$ |
| $\text{radius}(F, u)$ | The radius of $\text{KNN}(F, u)$ |
| $\text{RGN}^+ (u, F)$ | The positive region of $u \in U$ w.r.t. the query of $\text{RkNN}(U, F, q)$ |
| $\text{RGN}^- (u, F)$ | The negative region of $u \in U$ w.r.t. the query of $\text{RkNN}(U, F, q)$ |
| $\text{Vcell}(P, p)$ | The Voronoi diagram of the point set $P$ |
| $\text{Vneighbors}(P, p)$ | The Voronoi neighbors of point $p \in P$ in $\text{VD}(P)$ |
| $\text{DG}(P)$ | The Delaunay graph of the point set $P$ |
| $\text{U-RGN}^+ (U, F, q)$ | The positive united region of the query of $\text{RkNN}(U, F, q)$ |
| $\text{U-RGN}^- (U, F, q)$ | The negative united region of the query of $\text{RkNN}(U, F, q)$ |
| $\Delta p_1 p_2 p_3$ | A triangle with vertices at points $p_1$, $p_2$ and $p_3$ |
| $L_{p_1,p_2}$ | The line segment between points $p_1$ and $p_2$ |
| $\text{U-RG}^- (U, F, q)$ | The complementary region of $\text{U-RGN}^- (U, F, q)$.
any point on it to two fixed points (the foci) \( f_1 \) and \( f_2 \) is always equal to a constant (the length of the major axis) \( l \). Formally,

\[
E_{f_1,f_2} = \{ p | d(f_1, p) + d(f_2, p) = l \}. \tag{4}
\]

**Definition 3.5** (Hyperbola). As shown in Figure 2(c), a hyperbola (or a hyperboloid in a multi-dimensional space) \( \mathcal{H}_{f_1,f_2} \) is a geometric shape such that the difference between the distances from any point on it to two fixed points (the foci) \( f_1 \) and \( f_2 \) is always equal to a constant (the length of the major axis) \( l \). Formally,

\[
\mathcal{H}_{f_1,f_2} = \{ p | |d(f_1, p) - d(f_2, p)| = l \}. \tag{5}
\]

Next, four regions are defined based on the above three types of conic sections: the \( k \)NN region, the positive region, the negative region and the semi-\( k \)NN region. Using these four regions, we review a classical R\( k \)NN verification approach and propose three novel approaches to expedite the verification of R\( k \)NN candidates.

**Definition 3.6** (\( k \)NN region). The \( k \)NN region of a user \( u \) w.r.t. a facility set \( F \) is the region enclosed by \( C^{d(u,f_k)}_u \), where \( f_k \) denotes the \( k \)th nearest facility to \( u \) in \( F \). The radius of this region, \( d(u,f_k) \), is referred to as the \( k \)NN radius of \( u \) w.r.t. \( F \). For convenience, the \( k \)NN region and \( k \)NN radius of \( u \) w.r.t. \( F \) are denoted as \( RG_{kNN}(u) \) and \( r_{kNN}(F,u) \), respectively.

According to the definition of R\( k \)NN queries, Lemma 3.1 is apparently obtained.

**Lemma 3.1** (\( k \)NN discrimination). For a query of R\( k \)NN(\( U,F,q \)), if the query facility \( q \) lies in the \( k \)NN region of a user \( u \in U \) w.r.t. \( F \), then \( u \) is an R\( k \)NN user. Otherwise \( u \) is a non-R\( k \)NN user.

In other words, if \( d(u,q) \leq r_{kNN}(F,u) \), then \( u \in RkNN(U,F,q) \). Conversely, if \( d(u,q) > r_{kNN}(F,u) \), then \( u \notin RkNN(U,F,q) \).

**Example** (Lemma 3.1). As shown in Figure 3, \( q \) lies in both \( RG_{kNN}(u_1) \) and \( RG_{kNN}(u_2) \), but not in \( RG_{kNN}(u_3) \), \( RG_{kNN}(u_4) \) and \( RG_{kNN}(u_5) \). Thus, we can determine that both \( u_1 \) and \( u_2 \) belong to the R\( k \)NN set of \( q \), while \( u_3, u_4 \) and \( u_5 \) do not.

The expected size of the R\( k \)NN set has been proven to be \( k|U|/|F| \) (Cheema et al. 2011). The candidate set, being a superset of the R\( k \)NN set, cannot be smaller than this. Some classical algorithms, such as VR-R\( k \)NN, adopt Lemma 3.1 to verify...
candidates, with computational complexity of $O(\log |F| + k \cdot \log k)$ (Sharifzadeh and Shahabi 2010). Although there are already algorithms that offer relatively more efficient verification methods, it is still computationally expensive to perform such verification on the entire candidate set. To address this issue, we propose three more efficient verification methods, allowing most candidates to be verified with $O(1)$ computational complexity.

**Definition 3.7** (Positive region). For a query of $R_kNN(U, F, q)$, the positive region of an user $u \in U$, denoted as $RG_{F,q}^{+}(u)$, is the internal region of $C_{u,q}^{RkNN}(F)$. Formally, it is defined as follows:

$$RG_{F,q}^{+}(u) = \{ p | d(p, q) + d(p, u) \leq r_{kNN}(F, u) \}.$$  

(6)

**Lemma 3.2** (Ellipse discriminance). For a query of $R_kNN(U, F, q)$, if a user $u$ is located in the positive region of an $R_kNN$ user $u'$, i.e. Inequality (7) is satisfied, then $u$ must be an $R_kNN$ user.

$$d(u, q) + d(u, u') \leq r_{kNN}(F, u')$$  

(7)

**Example** (Lemma 3.2). As shown in Figure 4(a), $u_1$ and $u_2$ lie in $RG_{F,q}^{+}(u')$, then we know that both of them are RkNN users. Whereas $u_3$, $u_4$ and $u_5$ lie out of $RG_{F,q}^{+}(u')$, so whether they are RkNN users cannot be determined directly.

**Proof** (Lemma 3.2). As shown in Figure 4(b), for an arbitrary point $p$ located in $C_{u,q}^{RkNN}(F, u') - d(u, u')$, it holds that $d(u, p) \leq r_{kNN}(F, u') - d(u, u')$, which is equivalent to $d(u, p) + d(u, u') \leq r_{kNN}(F, u')$. By the triangular inequality, we have $d(p, u') \leq d(u, p) + d(u, u') \leq r_{kNN}(F, u')$, implying that $p$ is inside $C_{u,q}^{RkNN}(F, u')$. Hence, we can infer that $C_{u,q}^{RkNN}(F, u') - d(u, u')$ is contained in $C_{u,q}^{RkNN}(F, u')$. Therefore, the number of facilities in $C_{u,q}^{RkNN}(F, u') - d(u, u')$ cannot exceed $k$, which is the number of facilities in $C_{u,q}^{RkNN}(F, u')$. Equivalently, the number of facilities lying in $C_{u,q}^{RkNN}(F, u') - d(u, u')$ cannot be greater than

![Figure 3. Example diagram of Lemma 3.1.](image-url)
that in \(C_u^\text{NN}(F,u)\). Since these two circles are concentric, neither area nor radius of the former can be greater than that of the latter. Thus, we have \(r_{\text{NN}}(F,u') - d(u,u') \leq r_{\text{NN}}(F,u)\). As \(u\) lies in \(R_{F,q}(u')\), i.e. Inequality (7) holds, then \(d(u,q) \leq r_{\text{NN}}(F,u') - d(u,u') \leq r_{\text{NN}}(F,u)\). It follows Lemma 3.1 that \(u \in \text{RkNN}(U,F,q)\). \(\square\)

**Lemma 3.2** presents a sufficient yet unnecessary condition for a user \(u \in U\) to be an RkNN user of a query facility \(q \in F\). Note that it is not required to perform a kNN query for \(u\). When \(r_{\text{NN}}(F,u')\) is given, only using the Euclidean distances between \(u\) and \(q\) and that between \(u\) and \(u'\) can determine if Inequality (7) in Lemma 3.2 holds. It is an atomic operation that calculates the distance between two given points. The discriminance related to Lemma 3.2 only needs \(O(1)\) computational complexity.

**Definition 3.8** (Negative region). The space is separated by \(H_{q,u}^\text{NON}(F,u')\) into three parts. Among them, the one that \(u' \in U\) is located in is referred to as the negative region of \(u'\) w.r.t \(\text{RkNN}(U,F,q)\) and represented as \(R_{F,q}(u')\). In specific,

\[
R_{F,q}(u') = \{p|d(p,q)-d(p,u') > r_{\text{NN}}(F,u')\} \tag{8}
\]

**Lemma 3.3** (Hyperbola discriminance). For a query of \(\text{RkNN}(U,F,q)\), if a user \(u\) is located in the negative region of a non-RkNN user \(u'\), i.e. Inequality (9) is satisfied, then \(u\) cannot be an RkNN user.

\[
d(u,q) - d(u,u') > r_{\text{NN}}(F,u'). \tag{9}
\]

**Example** (Lemma 3.3). As shown in Figure 5(a), \(u_1, u_2\) and \(u_3\) lie inside \(R_{F,q}(u')\), while \(u_4\) and \(u_5\) do not. Then we can determine that neither \(u_1, u_2\) nor \(u_3\) belongs to \(\text{RkNN}(U,F,q)\). However, Lemma 3.3 is not able to help us decide whether \(u_4\) and \(u_5\) belong to \(\text{RkNN}(U,F,q)\).
Proof (Lemma 3.3). As shown in Figure 5(b), given an arbitrary point \( p \) in \( RGF^c_{kNN}(u'), d(u', p) \leq r_{kNN}(F, u') \) is satisfied. According to the triangular inequality, we have \( d(u, p) \leq d(u, u') + d(u', p) \leq d(u, u') + r_{kNN}(F, u') \) which means \( p \) is inside \( C^d_u(u, u') + r_{kNN}(F, u') \). Then, we can deduce that \( RGF^c_{kNN}(u') \) is contained in \( C^d_u(u, u') + r_{kNN}(F, u') \), i.e. there are at least \( k \) facilities in \( C^d_u(u, u') + r_{kNN}(F, u') \). Hence, the number of facilities in \( C^d_u(u, u') + r_{kNN}(F, u') \) cannot be less than that in \( RGF^c_{kNN}(u) \). Since \( C^d_u(u, u') + r_{kNN}(F, u') \) and \( RGF^c_{kNN}(u) \) are concentric, both the area and radius of the former must be greater than or equal to those of the later, i.e. \( d(u, u') + r_{kNN}(F, u') \geq r_{kNN}(F, u) \). Because \( u \) lies in \( RG_{F,q}(u') \), i.e. Inequality (9) holds, \( d(u, q) > d(u, u') + r_{kNN}(F, u') \geq r_{kNN}(F, u) \). From Lemma 3.1, \( u \subset RkNN(U, F, q) \) is deduced.

**Lemma 3.3** presents a sufficient yet unnecessary condition for a user being a non-RkNN user. Same as **Lemma 3.2**, the discriminance of **Lemma 3.3** requires a computational complexity as \( O(1) \) if the \( kNN \) radius of an appropriate reference user is already obtained.

**Definition 3.9** (Semi-kNN region). The semi-\( kNN \) region of a facility \( q \in F \) is the area enclosed by \( C^d_{kNN}(F, q)/2 \). It is formally represented as \( Semi-RGF^c_{kNN}(q) \), where

\[
Semi-RGF^c_{kNN}(q) = \{ p | d(p, q) < r_{kNN}(F, q)/2 \}.
\]

**Lemma 3.4** (Semi-\( kNN \) discriminance). If a user \( u \) lies in the semi-\( kNN \) region of a query facility \( q \) of an \( RkNN \) query, then \( u \) must belong to the \( RkNN \) set of \( q \).

**Example** (Lemma 3.4). As shown in Figure 6(a), \( u_1 \) and \( u_2 \) are located in \( Semi-RGF^c_{kNN}(q) \), while \( u_3, u_4 \) and \( u_5 \) are not. By **Lemma 3.4**, \( u_1 \) and \( u_2 \) belong to \( RkNN(U, F, q) \), whereas it is difficult to decide if \( u_3, u_4 \) and \( u_5 \) are members of \( RkNN(U, F, q) \).
Proof (Lemma 3.4). As shown in Figure 6(b), given an arbitrary point \( p \) in \( C_u^{d(u,q)} \), \( d(u,p) \leq d(u,q) \) is satisfied. It also, \( d(q,p) \leq d(u,q) + d(u,p) \leq 2 \cdot d(u,q) \) holds, based on the triangular inequality. As \( u \) lie in \( \text{Semi-RGF}_{k \text{NN}}(q) \), \( d(u,q) < r_{k \text{NN}}(F,q)/2 \) holds. As aforementioned, \( d(q,p) < r_{k \text{NN}}(F,q) \). Then we can deduce that \( C_u^{d(u,q)} \) is absolutely contained in \( \text{RGF}_{k \text{NN}}^F(q) \) (where ‘absolutely’ means that the former does not touch the boundary of the latter). Since there are exactly \( k \) facilities (including \( q \) but excluding the \( k \)th closest facility to \( q \)) absolutely lying inside \( \text{RGF}_{k \text{NN}}^F(q) \), there cannot be more than \( k \) facilities within \( C_u^{d(u,q)} \), which means \( d(u,q) \leq r_{k \text{NN}}(F,u) \). According to Lemma 3.1, \( u \) belongs to \( \text{RkNN}(U,F,q) \).

Once the \( k \text{NN} \) radius of \( q \) is calculated, we can use Lemma 3.4 to directly determine all the users in \( \text{Semi-RGF}_{k \text{NN}}^F(q) \) as \( \text{RkNN} \) users with \( O(1) \) computational complexity.

Unifying the aforementioned four lemmas, CSD is proposed as a complete \( \text{RkNN} \) verification method. With CSD, we can efficiently find all the \( \text{RkNN} \) users by performing \( k \text{NN} \) queries for only the reference users (which are only a small part of the candidates) and the query facility.

### 3.3. Voronoi-based candidate retrieval strategy

To help understand the mechanism of our pruning operation, we first introduce two important concepts: the Voronoi diagram and the Delaunay graph (Li and Liu 2020).

Definition 3.10 (Voronoi diagram). As shown in Figure 7(a), the Voronoi diagram of a point set \( P \), denoted as \( \text{VD}(P) \), is formed by the collection of Voronoi cells corresponding to each point in \( P \). Specifically, the Voronoi cell associated with a point \( p \in P \), denoted as \( V_{\text{cell}}(P,p) \), is the region in which any point has \( p \) as its closest point in \( P \). The Voronoi neighbors of \( p \), denoted as \( V_{\text{neighbors}}(P,p) \), consist of the points in \( P \) whose Voronoi cells intersect \( V_{\text{cell}}(P,p) \).
**Definition 3.11** (Delaunay graph). As shown in Figure 7(b), the Delaunay graph \( DG(P) \) of \( P \) is a connected graph dual to \( VD(P) \). In \( DG(P) \), each pair of Voronoi neighbors is directly connected by an edge.

This subsection focuses on utilizing Voronoi diagrams and Delaunay graphs to retrieve candidates. Our core idea is to identify potential RkNN users by traversing a connected sub-graph of \( DG(U) \). Therefore, the initial task of this subsection is to obtain a connected sub-graph that adequately encompasses all the RkNN users.

**Definition 3.12** (Positive (negative) united region). In the RkNN(\( U, F, q \)) query, the positive (negative) united region, denoted as \( U-RG^+(U,F,q) \) (\( U-RG^-(U,F,q) \)), refers to the collective area formed by the positive (negative) regions of all RkNN (non-RkNN) users. It is depicted as the shaded area bounded by red (green) curves in Figure 8(a).

**Definition 3.13** (Border user). For RkNN(\( U, F, q \)), if the line segment between a user \( u \in U \) and one \( V_{\text{neighbors}}(U,u) \) intersects the border \( U-RG^-(U,F,q) \), then \( u \) is a border user. As shown in Figure 8(b), since the yellow edges intersect the border of the negative united region (the red curve), their endpoints (vertices marked by yellow circles) are border users.
It can be observed in Figure 8(b) that the border \( U\text{-RG}^- (U,F,q) \) intersects multiple triangles in \( DG(U) \), and these triangles are adjacent to each other due to the shared edges (highlighted in yellow). Any two edges within the same triangle share a common endpoint, so the edges that intersect with the border of \( U\text{-RG}^- (U,F,q) \) are necessarily interconnected. Consequently, the sub-graph of \( DG(U) \) formed by these intersecting edges, along with their corresponding endpoints (i.e. border users) is a connected graph. Since \( DG(U) \) is a connected graph, each \( RkNN \) user within it is guaranteed to have at least one Voronoi neighbor. Additionally, as per the definition of border users, there is no direct connection in \( DG(U) \) possible between any \( RkNN \) user and non-\( RkNN \) user in the absence of any border users. Namely, any Voronoi neighbor of an \( RkNN \) user can only be an \( RkNN \) user or a border user. Thus, there exists in \( DG(U) \) at least one connected path between any \( RkNN \) user and any border user that exclusively passes through \( RkNN \) users and border users. As a result, we can conclude that the sub-graph of \( DG(U) \) consisting of all the border users, all the \( RkNN \) users, and the edges between them is a connected graph. So, a reasonable \( RkNN \) candidate set that contains all the \( RkNN \) users can be obtained by traversing this sub-graph starting with an \( RkNN \) user or a border user. Next, we present Lemma 3.5 to describe how to select the starting user for the traversal.

**Lemma 3.5.** Given the Delaunay triangle \( \triangle u_1u_2u_3 \) in \( DG(U) \) in which \( q \in F \) lies, if \( RkNN(U,F,q) \) is not empty, there must be at least one among \( u_1, u_2 \) and \( u_3 \) being either a border user or an \( RkNN \) user.

**Proof (Lemma 3.5).** Let \( \overline{U\text{-RG}^- (U,F,q)} \) be the complementary region of \( U\text{-RG}^- (U,F,q) \). If a user lies in the same location as \( q \) (i.e. they have the same coordinates), \( q \) must be its closest facility, and then it must be an \( RkNN \) user. So \( q \) cannot lie in \( U\text{-RG}^- (U,F,q) \), i.e. \( q \) must lie in \( \overline{U\text{-RG}^- (U,F,q)} \). Since \( q \) lies in \( \triangle u_1u_2u_3 \), \( \overline{U\text{-RG}^- (U,F,q)} \) intersects \( \triangle u_1u_2u_3 \). If \( RkNN(U,F,q) \) is not empty, there must exist at least one user lying in \( \overline{U\text{-RG}^- (U,F,q)} \). Because there are no other users in \( \triangle u_1u_2u_3 \) other than \( u_1, u_2 \) and \( u_3 \), \( \overline{U\text{-RG}^- (U,F,q)} \) cannot be completely inside \( \triangle u_1u_2u_3 \). Thus, there are only two possible relationships between \( \overline{U\text{-RG}^- (U,F,q)} \) and \( \triangle u_1u_2u_3 \): \( \odot \) \( \triangle u_1u_2u_3 \) lies completely inside \( \overline{U\text{-RG}^- (U,F,q)} \); \( \otimes \) \( U\text{-RG}^- (U,F,q) \) and \( \triangle u_1u_2u_3 \) are only partially overlapping. If the former case holds, \( u_1, u_2 \) and \( u_3 \) are \( RkNN \) users. If the latter case holds, the border of \( \triangle u_1u_2u_3 \) intersects the border of \( U\text{-RG}^- (U,F,q) \) (note that \( U\text{-RG}^- (U,F,q) \) and \( U\text{-RG}^- (U,F,q) \) share the same border), then among \( u_1, u_2 \) and \( u_3 \), there must be at least one being a border user. Thus, Lemma 3.5 is proven to be true.

As per Lemma 3.5, the vertices of the triangle where the query facility is located can serve as the initial candidates, and at least one of them can be chosen as the starting user for the aforementioned traversal. During this traversal, all the \( RkNN \) users and border users are required to access. The verification method for \( RkNN \) users is already described in the previous subsection. However, it is very difficult to accurately calculate the border of \( U\text{-RG}^- (U,F,q) \). Hence, it is impractical to determine whether a user is a border user or not by geometric methods. Thinking differently, we can regard this traversal as filtering out the users that are neither \( RkNN \) users nor border users. Then, we introduce Lemma 3.6, which provides a sufficient condition to identify them.
Definition 3.14 (Discriminant user). If a user $u$ lies in the positive or negative region of another user $u'$, it is stated that $u'$ is a positive or negative discriminant user of $u$, or that $u'$ can positively or negatively discriminate $u$.

Lemma 3.6. If a non-RkNN user can share a negative discriminant user with each of its own Voronoi neighbors, then it cannot be a border user.

Proof (Lemma 3.6). Let $u$ and $u_n$ be a user in $U$ and a Voronoi neighbor of $u$, respectively. Suppose that $u^- \in U$ is a user that can negatively discriminate both $u$ and $u_n$. Then, $u$ and $u_n$ both lie in $RG_{F,q}(u^-)$. Boyd and Vandenberghe (2014) show that a hyperbolic cone (a conical region enclosed by a hyperbola or hyperboloid) is a convex set (a set such that the line segment between any two points in it lies in it). Therefore, $RG_{F,q}(u^-)$ is a convex set, and $L_{u,u_n}$ (the line segment with $u$ and $u_n$ as endpoints) lies in it. Since $RG_{F,q}(U^-) \in U-RG^-(U,F,q), L_{u,u_n}$ lies in $U-RG^-(U,F,q)$, i.e. $L_{u,u_n}$ does not intersect the boundary of $U-RG^-(U,F,q)$. From the above, if a user can share a negative discriminant user with each of its Voronoi neighbors, the line between it and any of its Voronoi neighbors does not intersect the boundary $U-RG^-(U,F,q)$, then it cannot be a border user. □

Note that when pruning a user by Lemma 3.6, the discriminant users of it and its Voronoi neighbors must be obtained first. So in our approach, the generation of candidates is a dynamic process accompanied by verification. In other words, the operations of candidate generation and validation of our algorithm are performed synchronously and alternately in parallel. The specific algorithms are detailed in the next section.

4. Algorithms

Based on the methodology discussed in the previous section, we have developed an RkNN search algorithm called CSD-RkNN. As shown in Figure 9, the algorithm incorporates two key processes, namely candidate generation and verification, aligning with the two

Figure 9. A schematic diagram of CSD-RkNN.
core ideas: ‘Voronoi based candidate generation’ and ‘conic section discrimination’. The candidate generation process involves initializing a candidate set (based on Lemma 3.5) and continuously generating new potential R\(k\)NN objects to update the candidate set (based on Lemma 3.6). In the verification process, each candidate generated by the former process is validated (based on Lemma 3.1 to 3.4). It is worth noting that during the candidate set update, the generation of new candidates relies on the results obtained by the verification process. This distinguishes our algorithm from traditional R\(k\)NN approaches, as the generation and verification of candidates are not separate and consecutive processes, but rather interdependent and performed alternately. The following two subsections provide a detailed explanation of the specific steps and pseudocodes for the proposed algorithm, addressing both bichromatic and Mono-R\(k\)NN queries.

### 4.1. Answering Bi-R\(k\)NN queries

First, we describe the algorithm for Bi-R\(k\)NN queries. Our approach requires to use of a Voronoi diagram to generate the candidates, so both user points and facility points are indexed by the VoR-tree. The main pseudocode is shown as Algorithm 1. In this algorithm, the candidates are maintained in a max-heap \(H_{\text{cnd}}\) by their distance from the query facility \(q\). First, we find the initial candidates, put them into \(H_{\text{cnd}}\), and mark them as visited (line 3 to line 5). Then, the users are iteratively popped from \(H_{\text{cnd}}\) and verified whether as R\(k\)NNs or not until \(H_{\text{cnd}}\) is empty (line 8 to line 17). During each iteration, unless the popped user is neither an R\(k\)NN user nor a border user, its Voronoi neighbors, which have not yet been visited, are placed into \(H_{\text{cnd}}\). Finally, the algorithm returns all the users, which are verified as R\(k\)NN users (line 18).

```
Input: U - user set; F - facility set; q - query facility; k - query parameter.
Output: R\(k\)NN set.
1 \(S_{\text{RNN}}\) \leftarrow Set();
2 \(H_{\text{cnd}}\) \leftarrow MaxHeap(); \(S_{\text{est}}\) \leftarrow Set();
3 \textbf{for} \(u \in \text{getInitialCandidates}(U, q)\) \textbf{do}
4 \hspace{1em} \(H_{\text{cnd}}\).push([d(u, q), u]);
5 \hspace{1em} \(S_{\text{est}}\).add(u);
6 \hspace{1em} \(D_{\text{dsc}}^+ \leftarrow \text{Dict}(); \hspace{0.5em} D_{\text{dsc}}^- \leftarrow \text{Dict}(); \hspace{0.5em} D_\tau \leftarrow \text{Dict}();
7 \hspace{1em} D_\tau[q] \leftarrow r_{\text{NN}}(F, q);
8 \textbf{while} \(H_{\text{cnd}}\).size > 0 \textbf{do}
9 \hspace{1em} \text{dist}_u, u \leftarrow H_{\text{cnd}}.pop();
10 \hspace{1em} \textbf{if} \text{isRNN}(F, q, k, D_{\text{dsc}}^+, D_{\text{dsc}}^-, D_\tau, u) \textbf{then}
11 \hspace{2em} S_{\text{RNN}}.add(u);
12 \hspace{1em} \text{else if} \text{cannotBeBorderUser}(F, q, k, D_{\text{dsc}}^+, D_{\text{dsc}}^-, D_\tau, u) \textbf{then}
13 \hspace{2em} \text{continue};
14 \hspace{1em} \textbf{foreach} \(u_n \in V_{\text{neighbors}}(U, u)\) \textbf{do}
15 \hspace{2em} \textbf{if} \(u_n \notin S_{\text{est}}\) \textbf{then}
16 \hspace{3em} \(H_{\text{cnd}}\).push([d(u_n, q), u_n]);
17 \hspace{3em} S_{\text{est}}.add(u_n);
18 \textbf{return} S_{\text{RNN}};
```
The function of initializing the candidate set is shown in Algorithm 2. The vertices of the Delaunay triangle in which the query facility \( q \) lies are taken as the initial candidates (see Lemma 3.5). We temporarily refer to this triangle as the target triangle for short. In most cases, the target triangle takes \( u^* \), the closest user to \( q \), as one of its vertices, but occasionally there are exceptions. In order to ensure robustness, we design a strategy to find the target triangle by traversing the Delaunay graph of the users starting from \( u^* \). First, we issue an NN (nearest neighbor) query to find \( u^* \) (line 1). Then all the triangles which take \( u^* \) as one of their vertices are placed in a queue \( Q_{trg} \) and marked as visited (line 3 to line 4). Later, the triangles are iteratively popped from \( Q_{trg} \) until \( Q_{trg} \) is empty (line 5 and line 6). Once a popped triangle \( t \) is found with \( q \) lying inside, the vertices of \( t \) are returned (line 7 and line 8). Otherwise, the triangles that share vertices with \( t \) and have not yet been visited, are put into \( Q_{trg} \) and marked as visited (line 9 to line 13). Even if the target triangle does not have \( u^* \) as its vertex, it will be very close to \( u^* \). Thus, in general, the above iteration will be executed only a few times. Hence, we can estimate the complexity of Algorithm 2 to be \( O(1) \).

---

**Input:** \( U \) - user set; \( q \) - query facility.

**Output:** initial candidates.

1. \( u^* \leftarrow \text{NN}(U, q) \);
2. \( Q_{trg} \leftarrow \text{Queue}() ; S_{vst} \leftarrow \text{Set}() ;
3. \text{for } t \in u^*:\text{triangles do}
   4. \quad \text{Q}_{trg}.\text{add}(t) ; \ S_{vst}.\text{add}(t) ;
5. \text{while } Q_{trg}\text{.size} > 0 \text{ do}
   6. \quad t \leftarrow Q_{trg}.\text{pop}() ;
6. \quad \text{if } q \text{ lies in } t \text{ then}
   7. \quad \quad \text{return } t:\text{vertices} ;
   8. \quad \text{else}
   9. \quad \quad \text{for } v \in t:\text{vertices do}
   10. \quad \quad \quad \text{for } t' \in v:\text{triangles do}
   11. \quad \quad \quad \quad \text{if } t' \notin S_{vst} \text{ then}
   12. \quad \quad \quad \quad \quad \text{Q}_{trg}.\text{add}(t') ; \ S_{vst}.\text{add}(t') ;

The verification function is defined as Algorithm 3. The algorithm uses three dictionaries, \( D^+_{dsc}, D^-_{dsc} \) and \( D_r \), to store the positive and negative discriminant users and the \( k\)NN radius for each user, respectively. First, we try to verify the user \( u \) with Lemma 3.4. If \( u \) lies in \( \text{Semi-RGF}_{kNN}(q) \), the algorithm returns true directly (line 1 and line 2). Otherwise, we utilize Lemma 3.2 or Lemma 3.3 to verify \( u \). If a Voronoi neighbor \( u_o \) of \( u \) has been verified as an RkNN user and its positive discriminant user \( u^+ \) can positively discriminate \( u \), then \( u^+ \) is saved into \( D^+_{dsc} \) as the positive discriminant user of \( u \) and true is returned (line 4 to line 8). If \( u_n \) has been verified as a non-RkNN user and its negative discriminant user \( u^- \) can negatively discriminate \( u \), then \( u^- \) is saved into \( D^-_{dsc} \) as the negative discriminant user of \( u \) and false is returned (line 9 to line 13). If none of Lemma 3.2, Lemma 3.3 or Lemma 3.4 can determine whether \( u \) is an RkNN user or not, then we use Lemma 3.1 to verify \( u \). If \( q \) is located in the \( k\)NN region of \( u, u \) is saved into \( D^+_{dsc} \) as the positive discriminant user of itself, and true is
returned (line 15 to line 17). Otherwise, \( u \) is saved into \( D_{dsc}^- \) as the negative discriminant user of itself, and false is returned (line 18 to line 20).

\[
\text{Algorithm 4 is used to identify the non-RkNN users who cannot be classified as border users. According to Lemma 3.6, if every Voronoi neighbor of } u \text{ is a non-RkNN user and shares the negative discriminant user with } u, u \text{ must not be a border user. How to determine whether two non-RkNN users } u_1 \text{ and } u_2 \text{ share the negative discriminant user is described in Algorithm 5. First, the negative discriminant users } u_1^- \text{ and } u_2^- \text{ of } u_1 \text{ and } u_2 \text{ are obtained respectively (line 1). If } u_1^- \text{ and } u_2^- \text{ represent the same user, true is returned (line 2 and line 3). If } u_1^- \text{ can negative discriminate } u_2 \text{ or } u_2^- \text{ can negative discriminate } u_1, \text{ true is returned (line 4 and line 5). If neither of the above two conditions is satisfied, the algorithm returns false (line 6).} \]

\[
\text{Input: } F - \text{ facility set; } q - \text{ query facility; } k - \text{ query parameter; } D_{dsc}^+ - \text{ positive discriminant user dictionary; } D_{dsc}^- - \text{ negative discriminant user dictionary; } D_r - k\text{NN radius dictionary; } u - \text{ a user to be verified.} \\
\text{Output:} \text{ whether } u \text{ belongs to the RkNNs of } q.
\]

1. if \( d(u, q) < D_r[q]/2 \) then /* Lemma 3.4 */
   2. return true;
3. foreach \( u_n \in V_{neighbors}(U, u) \) do
   4. if \( u_n \in D_{dsc}^+\text{-keys} \) then /* Lemma 3.2 */
   5. \( u^+ \leftarrow D_{dsc}^-[u_n]; \)
   6. if \( d(u, q) + d(u, u^+) \leq D_r[u^+] \) then
   7. \( D_{dsc}^+[u] \leftarrow u^+; \)
   8. return true;
   9. if \( u_n \in D_{dsc}^-\text{-keys} \) then /* Lemma 3.3 */
   10. \( u^- \leftarrow D_{dsc}^-[u_n]; \)
   11. if \( d(u, q) - d(u, u^-) > D_r[u^-] \) then
   12. \( D_{dsc}^-[u] \leftarrow u^-; \)
   13. return false;
14. \( D_r[u] \leftarrow r_{kNN}(F, u); \)
15. if \( D_r[u] \geq d(u, q) \) then /* Lemma 3.1 */
16. \( D_{dsc}^+[u] \leftarrow u; \)
17. return true;
18. else
19. \( D_{dsc}^-[u] \leftarrow u; \)
20. return false;
4.2. Answering Mono-RkNN queries

As mentioned above, Mono-RkNN is a special case of Bi-RkNN of which $U = F$. The Bi-RkNN algorithm described in the previous subsection requires only a few adjustments to answer Mono-RkNN queries:

- Because the closest one to $f$ in $F$ is $f$ itself, the $k$NN($F, f$) only contains $k - 1$ facilities, except for $f$. Therefore, when calculating $D_r[q]$, the parameter $k$ needs to be changed to $k + 1$, i.e. $D_r[q] \leftarrow r_{k+1} \text{NN}(F, q)$ (Algorithm 1 line 7).
- Since the query facility $q$ cannot lie in the negative discriminant region of any facility, $q$ cannot lie in $U-RG^-(F, F, q)$. Therefore, if a Voronoi neighbor of $q$ is not an RkNN facility, the line segment between $q$ and it must intersect the border of $U-RG^-(F, F, q)$, i.e. it must be a border facility. Then, we conclude that every Voronoi neighbor of $q$ must be either an RkNN facility or a border facility. Hence, instead of using Algorithm 2 to generate the initial candidate set, we directly take the Voronoi neighbors of $q$ as the initial candidates (Algorithm 1 line 3).
- The query facility $q$ cannot be one of the RkNNs of its own. Hence in Algorithm 3, if the facility to be verified is equal to $q$, false is directly returned.
5. Theoretical analyses

This section is aimed to scrutinize CSD-RkNN in terms of the expected candidate size and the expected number of discriminant users. Also, we intend to analyze the I/O cost and computational complexity of CSD-RkNN. It has been shown that the spatial distribution of the data has a certain impact on the performance of RkNN search techniques (Cheema et al. 2011, Yang et al. 2014, 2015, 2017). However, no method has emerged to accurately metric this impact so far. Hence, we can only discuss the theoretical average performance of our RkNN algorithm under uniform distributions. We assume that the facilities in \( F \) and the users in \( U \) are absolutely uniformly distributed in unit space. In the space, any two regions with the same area have the same number of users (or facilities) inside, every Voronoi cell of \( VD(U) \) has the same area \( \mathcal{A}_{8,U} \), every Delaunay triangle of \( DG(U) \) has the same area \( \mathcal{A}_{D,U} \), and the distance between any pair of Voronoi neighbor users is equal to the same constant \( d_U \). The sizes of \( U \) and \( F \) are \( |U| \) and \( |F| \), respectively.

5.1. Numbers of candidates and discriminant users

As the users in \( U \) are absolutely uniformly distributed in the space, every vertex (user) in \( DG(U) \) is shared by six Delaunay triangles. While any Delaunay triangle has three vertices, in other words, every Delaunay triangle is shared by three vertices (users). Therefore, the number of Delaunay triangles in \( DG(U) \) is \( \frac{|U|}{6/3} = \frac{2|U|}{3} \), and then \( \mathcal{A}_{D,U} \) can be calculated as \( \frac{1}{3|U|} \). As mentioned above, the distance between any pair of Voronoi neighbor users is \( d_U \), so each Delaunay triangle must be an equilateral triangle with sides of length \( d_U \). Then we obtain that \( \mathcal{A}_{D,U} = \sin \frac{\pi}{3} d_U^2 \), from which \( d_U = \sqrt{\frac{1}{3|U|}} \) is derived. It has been shown that the expectation of the size of the RkNN set \( k|U|/|F| \) and the expected area of the influence zone (which can be thought of as a region where only RkNN users are distributed) is \( k/|F| \). The authors of InfZone suggest that the larger the value of \( k \) is, the more the influence zone approximates a circle, and the radius of this circle \( r_k \) can be computed as \( \sqrt{\frac{k}{|F|}} \). So, we estimate the influence zone approximately as a circular region with a radius \( r_k = \sqrt{\frac{k}{|F|}} \). Our algorithm considers the RkNN users and border users as the candidates. Furthermore, a border user is generally either an RkNN user or a Voronoi neighbor of an RkNN user. Therefore, the candidate set \( S_{cnd} \) is almost equivalent to the set of RkNN users and their Voronoi neighbors. The distance from any RkNN user to the query facility \( q \) is less than or equal to \( r_k \), so the distance from any border user to \( q \) cannot be greater than \( r_k + d_U \). Hence, the area of the candidate region (i.e. the region that the candidates are distributed in) is about \( \pi(r_k + d_U)^2 \), and the expected number of candidates \( |S_{cnd}| \) can be estimated by Equation (11).

\[
|S_{cnd}| = \pi(r_k + d_U)^2|U| = k \left| \frac{|U|}{|F|} \right| + 2 \sqrt{k \frac{\pi|U|}{\sqrt{3}|F|}} + \frac{\pi}{\sqrt{3}} \tag{11}
\]
Our algorithm takes part of the candidates as the discriminant users to discriminate against the other candidates. Theoretically, it is sufficient to consider the border users as the discriminant users. In practice, our algorithm obtains the candidates one by one by traversing the sub-graph of $DG(U)$. During this traversal, the candidates are maintained in a max-heap by their distance from $q$, and the discriminant users are selected on demand from the candidates according to the greedy strategy. The max-heap allows the users further away from $q$ to be accessed in preference. That means the border users are preferred as discriminant users. In general, if an $R_kNN$ user is a border user, then it has at least one Voronoi neighbor that is not an $R_kNN$ user. Since any user that is less than $r_k/C_0d_U$ away from $q$ cannot have a non-$R_kNN$ user as a Voronoi neighbor, the distance from any border user cannot be less than $r_k/C_0d_U$.

Thus, the border users are distributed in an annular region that takes $r_k/C_0d_U$ and $r_k+2d_U$ as small and large radii, respectively. Therefore, the area of this region is $\pi[(r_k+d_U)^2-(r_k-d_U)^2] = 4\pi r_k d_U$, and the number of discriminant users $|S_{dsc}|$ can be estimated by Equation (12).

$$|S_{dsc}| = 4\pi r_k d_U|U| = 4\sqrt{k \pi|U|/\sqrt{3}|F|}$$

(12)

5.2. I/O cost

In total, the I/O cost is incurred by accessing the user index and the facility index. Among the users, only the candidates and their Voronoi neighbors are accessed. Because the distance from any candidate user to $q$ is not greater than $r_k+2d_U$, the distance from any accessed user to $q$ cannot be greater than $r_k+2d_U$. That means the area of the accessed region (i.e. the region where the accessed users are distributed) is $\pi(r_k+2d_U)^2$. Let $c$ be the leaf node size of the VoR-tree. Then, the I/O cost incurred by accessing the user points can be calculated as $\pi(r_k+2d_U)^2|U|/c$. During the generation of the initial candidates, the NN query must be executed once, so several additional intermediate nodes of the VoR-tree need to be accessed. Let $\Phi(|U|)$ represent the I/O cost of issuing an NN query on the VoR-tree of $U$. For the specific calculation approach, please refer to the existing work by Hjaltason and Samet (1999). Then, the I/O cost incurred by accessing the user index can be estimated by

$$\pi(r_k+2d_U)^2|U|/c + \Phi(|U|).$$

(13)

For our algorithm, the $kNN$ facilities of every discriminant user are accessed. Because the $kNN$ region of a user must contain $k$ facilities, its area is $k/|F|$ and its radius is $\sqrt{k/|F|}$, which is equal to $r_k$. So for any accessed facility, the distance from its nearest discriminant user cannot be greater than $r_k$, and the distance from $q$ cannot be greater than $2r_k+d_U$. In other words, the area of the region where the accessed facilities are distributed is $\pi(2r_k+d_U)^2$. For a VoR-tree, the number of intermediate nodes accessed by the $kNN$ query is equal to that of the NN query (Sharifzadeh and Shahabi 2010). Hence, the number of intermediate nodes accessed by our algorithm cannot be greater than $\Phi(|F|)|S_{end}|$. Therefore, the upper bound of the I/O cost incurred by accessing the facility index can be estimated by
5.3. Computational complexity

With the help of an R-tree-derived index (e.g. VoR-tree), the NN of any point can be found in the logarithmic time (Hjaltason and Samet 1999). Thus, the computational complexity of generating the initial candidates is $O(\log |U|)$. Once the initial candidates are obtained, the generation and verification of the other candidates are carried out synchronously and alternately. In this phase, it is only necessary for the discriminant users to be verified using $k$NN queries, whereas every other candidate can be verified with $O(1)$ complexity. The VoR-tree-based $k$NN query demands complexity $O(\log n + k \log k)$, where $n$ is the number of points indexed by the VoR-tree (Sharifzadeh and Shahabi 2010). Therefore, the expected computational complexity of our algorithm can be estimated by

$$\pi(2r_k + d_U)^2|F|/c + \Phi(|F|)|S_{cnd}|$$  \hspace{1cm} (14)

$$O(|S_{disc}|(\log |F| + k \log k) + |S_{cnd}| + \log |U|)$$

$$= O(\sqrt{|U|/|F|}(\sqrt{k} \log |F| + k^{1.5} \log k) + \log |U|).$$  \hspace{1cm} (15)

In practice, the user set and the facility set are generally fixed, allowing $|U|$ and $|F|$ to be treated as constants. In this case, the time complexity of CSD-RkNN is $O(k^{3/2} \log k)$ that is significantly lower than the $O(k^2)$ of SLICE and the $O(k^2 \log k)$ of VR-RkNN.

6. Experiments and evaluations

This section evaluates the performance of our algorithm through a series of comparative experiments, including two parts: benchmark experiments and case study experiments. In each set of our experiments, the proposed algorithm CSD-RkNN (referred to as CSD) is compared with VR-RkNN (referred to as VR) and SLICE. VR is the original RkNN algorithm implemented on VoR-tree, and SLICE is one of the state-of-the-art algorithms in this domain.

6.1. Experimental environment settings

To ensure algorithm readability, experiment fairness and code extensibility, all algorithms were implemented in Python 3.5. Additionally, all experiments were conducted on the same personal computer with the following specifications: Intel Core i5-4308U 2.80 GHz CPU, 8GB DDR3 RAM and macOS Catalina 10.15.5 operating system. Since both VR and CSD need to take advantage of Voronoi diagrams, all data points are organized by VoR-tree, where the page size is 4,096 bytes. It is worth noting that for SLICE, there is no difference between VoR-tree and R-tree in terms of data retrieval, as it does not use the Voronoi diagram component of VoR-tree. To improve precision, each experiment is repeated 30 times, and the average results were computed.
6.2. Experimental data descriptions

In the benchmark experiments, we employed the same experimental data as our main competitor, SLICE, encompassing both real and synthetic data sets. The real data set consists of 175,803 locations in North America, divided into two sets of almost equal sizes representing the facilities and users. The default synthetic data set consists of 100,000 randomly generated points distributed either normally or uniformly. Figure 10 illustrates the point distributions of the benchmark experimental data.

In the case study experiments, Wuhan, the capital of Hubei Province in China, which is rich in education, science, healthcare and business, was selected as the study area. A total of 102,315 POIs (points of interest) of five types in Wuhan were used in the experiments: 7375 schools (including kindergartens, primary schools, middle schools and colleges), 7487 hospitals (including general hospitals, specialized hospitals, health centers, emergency centers and clinics), 678 malls (including general malls, shopping centers, shopping plazas, supermarkets and home furnishing markets), 71,474 restaurants (including Chinese restaurants, foreign restaurants and fast-food restaurants) and 15,301 residences (residential districts, dormitories and villas). The distributions of the five types of POIs of the case study experiments data are shown in Figure 11.

6.3. Benchmark experiments

This subsection studies the effect of data size, the value of $k$, the ratio of the number of users to the number of facilities, and the data distribution on the performance of R$k$NN algorithms through the benchmark experiments. To be fair, we adopt similar experimental settings to those used in SLICE. For Bi-R$k$NN queries, the user set and the facility set have the same size and follow the same distribution by default, unless explicitly stated otherwise. The main purpose of our algorithm is to make up for the low performance of the existing R$k$NN algorithms in the case of large $k$ values. Therefore, on the basis of SLICE’s original experiments where the default value of $k$ is 10, we add experiments with large default $k$ values. Then, the default values of $k$ are 10 and 1000. The specific parameter configurations for each benchmark experiment group can be found in Table 2.

6.3.1. Effect of data size

In this set of experiments, the size of the synthetic data set varies evenly from 50,000 to 200,000. The real data used in the experiments were obtained from a fixed data set of size 175,803. We divided this data set into two equal parts representing the user set and the facility set, respectively. Following the approach described in reference (García-García et al. 2018), we randomly sampled the data from these two parts in proportions of 25%, 50%, 75% and 100% to construct four sets of real data that varied evenly in size from small to large. Figures 12 and 13 depict the impact of data size on the algorithms for Bi-R$k$NN and Mono-R$k$NN, respectively. The figures indicate that as the data set size increases, both the time cost and I/O cost of the algorithms exhibit a certain upward trend. However, the change in time cost appears to be more gradual. This is because the major computational overhead in R$k$NN queries stems from the
verification phase of the algorithms, and the computational complexity of the verification phase, in the three experimental algorithms, is dependent on the value of $k$. In terms of time cost, regardless of the data set size, CSD consistently outperforms the other two algorithms. In the experiment with $k = 10$, VR demonstrates lower time cost compared to SLICE. However, in the experiments with $k = 1000$, SLICE surpasses VR. This is attributed to VR’s sensitivity to the value of $k$ among the three algorithms, resulting in a significant increase in time cost as $k$ increases. In terms of I/O cost, the differences among the three algorithms are not significant when $k = 10$. However, when $k = 1000$, CSD exhibits the best performance, followed by SLICE, while VR performs the worst.

### 6.3.2. Effect of $k$

This set of experiments studies the effect of $k$ on RkNN algorithms through Figures 14 and 15. From the figures, it is evident that as the value of $k$ varies from 1 to 1000, the time and I/O costs of the three algorithms increase significantly. The overall computational complexities of CSD, SLICE and VR are $O(k^{3/2} \log k)$, $O(k^2)$ and $O(k^2 \log k)$, respectively. Therefore, regarding the sensitivity of time costs to the value of $k$, CSD exhibits lower sensitivity than SLICE, and SLICE shows lower sensitivity than VR. Unlike CSD and VR, SLICE requires computing several sigLists during the pruning phase, resulting in a computational complexity of $O(k^2)$, which is on the same order as the computational complexity of its verification phase. This computational overhead can be neglected only when the value of $k$ is sufficiently large. Therefore, as observed in the figures, when the value of $k$ is relatively small ($k \leq 100$), SLICE exhibits the highest overall time cost among the three algorithms. CSD effectively limits the size of the candidate set through the Voronoi-based candidate generation approach, resulting in significantly lower I/O cost compared to the other two algorithms. SLICE also reduces the candidate number through effective techniques, making its theoretical I/O cost lower than VR. However, VR leverages the Voronoi diagram to minimize access to intermediate nodes in the index. Therefore, when $k$ has a small value, VR exhibits lower I/O cost compared to SLICE.

![Figure 10](image.png)

**Figure 10.** Benchmark experimental data: points that follow (a) uniform distribution, (b) normal distribution and (c) real distribution.
6.3.3. Effect of the number of users relative to the number of facilities

This set of experiments evaluates the effect of $|U|$ relative to $|F|$, where $|F|$ is fixed at $100,000$ and $|U|$ varies from $25,000$ to $400,000$, i.e. $|U|/|F|$ varies from $25\%$ to $400\%$. As aforementioned, the expected size of the R$k$NN set is $k|U|/|F|$. Therefore, a larger $|U|/|F|$ implies that more users will be accessed, resulting in higher I/O costs. Thus, the overall trend reflected in Figure 16 is that both time cost and I/O cost rise as $|U|/|F|$ increases for all three algorithms. However, it is worth noting that when $k = 10$, the

Table 2. Parameter settings of the benchmark experiments.

| Experiments                     | Data size$^a$                      | $k$     | Distribution$^b$ |
|--------------------------------|------------------------------------|---------|-----------------|
| Effect of data size            | Uniform/normal: 50,000, 100,000, 150,000, 200,000; real: 21,975, 43,951, 65,926, 87,901 | 10, 1000 | U, N, R         |
| Effect of $k$                  | Uniform/normal: 100,000; real: 87,901 | 1, 10, 100, 1000 | U, N, R       |
| Effect of $|U|/|F|$             | (25,000, 100,000), (50,000, 100,000), (100,000, 100,000), (200,000, 100,000), (400,000, 100,000) | 10, 1000 | U, N           |
| Effect of distribution         | 87,901                             | 10, 1000 | (U,U), (U,R), (U,N), (R,U), (R,R), (R,N), (N,U), (N,R), (N,N) |

$^a$The value pairs in parentheses represent the sizes of the user set and the facility set, e.g. (25,000, 100,000) means that the sizes of the user set and the facility set are 25,000 and 100,000, respectively.

$^b$U, N and R stand for uniform, normal and real distributions, respectively. The character pairs in the parentheses stand for the distributions followed by the user set and the facility set, e.g. (U, N) means that the user set and the facility set follow uniform and normal distributions, respectively.

Figure 11. Case study experimental data: POIs of (a) residences, (b) malls, (c) restaurants, (d) hospitals and (e) schools.
I/O cost $|U|/|F| = 25\%$ is slightly higher than that at $|U|/|F| = 50\%$. This is because when the value of $k$ is very small, such as $k = 10$, a very small value $|U|/|F|$ will hardly reduce the I/O cost of accessing the user set. Instead, it increases the number of facilities that need to be accessed during the pruning and verification phases, resulting in higher I/O costs. Comparing the performance of the three algorithms, regardless of whether $k$ is small or large, VR is most sensitive to the value of $|U|/|F|$, followed by SLICE, while CSD is least sensitive.

6.3.4. Effect of data distribution

In this set of experiments, the effect of data distribution on the $R_k$NN algorithms is studied. Figure 17 shows the trends of time cost and I/O cost of the three experimental algorithms with various combinations of distributions. To be consistent with the real dataset in terms of size, each user set contains 87,901 points and each facility set contains 87,902 points, regardless of the distribution. From the experimental results, it can be seen that the distribution of data has a certain impact on $R_k$NN algorithms. In
the case of $k = 10$, CSD is similar to its competitors in terms of I/O cost, but its time cost is significantly lower than the other two algorithms. When $k = 1000$, the time cost and I/O cost of CSD are both much lower than those of the other two algorithms for each combination of data distributions.

### 6.4. Case study experiments

This section evaluates the performance of R$k$NN queries in real-world applications through a set of case study experiments in Wuhan. This set of experiments includes four scenarios, in which the potential clients of malls, restaurants, hospitals and schools are analyzed through R$k$NN queries, respectively. For privacy reasons, we use the POIs of residences to approximate the residents living in them. Assuming that the difference in the number of residents in each residence is not significant, it can be generally concluded that for facilities such as malls, restaurants, hospitals or schools, a larger number of R$k$NN residents correspond to a greater potential client base.

Figures 18 and 19 illustrate the effect of $k$ on the time cost and I/O cost of R$k$NN algorithms in the application of potential clients analysis in Wuhan, respectively. It can be observed from the results that no matter whether the query object is a mall, a restaurant, a hospital or a school, CSD has the best performance among the three experimental algorithms in terms of time cost. The larger $k$ is, the greater advantages CSD has over the other two algorithms. When $k$ reaches 256, CSD only needs computational time as $5.4–15.2\%$ of that of VR and $5.4–17.7\%$ of that of SLICE. In other words, the speed of CSD is $6.6–18.4$ times and $5.6–18.6$ times that of VR and SLICE, respectively. In terms of I/O cost, there is no significant difference between the three algorithms when $k$ is relatively small. However, with the increase of $k$, the I/O cost of CSD becomes significantly lower than that of the other two algorithms. When $k$ reaches 256, CSD’s I/O cost is only $27.2–74.7\%$ that of VR and $23.7–66.9\%$ that of SLICE. It is evident that CSD demonstrates superior performance in the R$k$NN query performed on
the four types of facilities. However, determining which is better between VR and SLICE depends on the specific scenario. Based on the findings discussed in Section 6.3.3, we can conclude that when the value of $|U|/|F|$ is small, VR performs better than SLICE, whereas when this value is large, SLICE exhibits better performance. Therefore, for facilities with a significantly large number of entities like restaurants, VR is more appropriate, while for facilities with a relatively smaller number of entities like malls, SLICE is recommended. According to the discussion in Section 6.3.2, VR is generally faster than SLICE for small values of $k$. However, as the value of $k$ increases, VR gradually becomes slower than SLICE. Therefore, for facilities with a moderate number of entities such as schools and hospitals, VR is more suitable when $k$ is small, while SLICE is more suitable when $k$ is large.

6.5. Experimental summary

From the experimental results presented in the previous two subsections, it can be concluded that CSD generally outperforms both VR and SLICE in terms of both time and I/O costs. Furthermore, CSD exhibits lower sensitivity to both the value of $k$ and the ratio $|U|/|F|$ compared to the other two algorithms. This means that regardless of

![Figure 16. Effect of the number of users relative to the number of facilities.](image)

![Figure 17. Effect of data distribution. The entries below the horizontal axis of each sub-graph represent the distributions of users and facilities, where U, N and R stand for uniform, normal and real distributions, respectively. For example, (U,N) means the data of users and facilities follow uniform and normal distributions, respectively.](image)
the chosen value of $k$ or the variation in the ratio $|U|/|F|$, CSD can maintain consistently high performance.

7. Conclusions and future works

In conclusion, CSD is proposed as a verification approach. It uses a few reference points (discriminant users) to determine whether the other candidates belong to the R$k$NN set. Furthermore, we propose a Voronoi-based candidate generation method to curtail the candidate set size. Based on VoR-tree, we combine these two approaches to form a novel R$k$NN algorithm, termed CSD-R$k$NN. The experimental study demonstrates that CSD-R$k$NN outperforms VR-R$k$NN and SLICE. More specifically, when $k$ is relatively small, CSD-R$k$NN is faster than the other two algorithms, and when facing large-scale R$k$NN queries, it has significant advantages both in time cost and I/O cost. The main contributions of this paper can be summarized in the following aspects:

- From a technical standpoint, CSD-R$k$NN encompasses two innovative techniques. First, we present the CSD verification method, which leverages simple geometric principles to significantly enhance the efficiency of the verification phase of R$k$NN algorithms. Second, we employ the Voronoi-based candidate generation method, effectively controlling the size of the R$k$NN candidate set and substantially reducing the I/O cost of the R$k$NN algorithm. These technological advancements not only improve the performance of R$k$NN queries but also offer new insights and technical references for the enhancement and development of other spatial query methods.

- From a practical application perspective, the CSD-R$k$NN algorithm addresses the real-world challenge of efficiently handling large $k$-value R$k$NN queries within tight time constraints, which existing techniques struggle to accomplish. This advancement

![Figure 18. Time cost (in seconds) of case study experiments.](image)

![Figure 19. Number of I/Os of case study experiments.](image)
enables RkNN queries, which have traditionally been time-consuming, to be applied in real-time spatial analysis domains with stringent responsiveness requirements. Moreover, it opens up new possibilities for the application of RkNN query techniques in diverse fields.

However, there are still aspects of the CSD-RkNN that warrant further research and improvement. Based on our current study, we plan to pursue the following works in the future:

- CSD-RkNN is primarily designed for snapshot RkNN queries in instantaneous scenarios. In the future, we aim to use CSD verification techniques to develop RkNN query algorithms for continuous scenarios, enabling spatial analysis over continuous time intervals in dynamic datasets.
- Currently, CSD-RkNN primarily considers the Euclidean distance metric. Our future work can extend the algorithm to support various distance metrics (e.g. Manhattan distance or cosine similarity), making it applicable to a wider range of domains and applications.
- While CSD-RkNN has demonstrated promising results in theoretical analysis and experimental evaluations, its real-world applicability needs to be explored. Future research will involve applying CSD-RkNN to practical domains, such as location-based services, recommendation systems or social network analysis. This will provide valuable insights into the algorithm’s performance, scalability and usability in real-world environments.

Acknowledgements

We sincerely thank the editors and the four anonymous reviewers for their useful comments and suggestions that significantly strengthened this manuscript.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the National Natural Science Foundation of China (Grant No. 42171466 and Grant No. U1711267); and the Humanities and Social Science Fund of Ministry of Education of China (Grant No. 22YJC630107).

Notes on contributors

Yang Li is currently a post-doctor researcher at the School of Geography and Information Engineering, China University of Geosciences. His research interests include spatial indexing, spatial queries and evolutionary computation.

Mingyuan Bai is currently a post-doctor researcher at Tensor Learning Team, RIKEN AIP. Her research interests include adversarial defense, tensor learning and generative models.
Qingfeng Guan is a professor at the School of Geography and Information Engineering, China University of Geosciences. His research interests include spatial intelligence, parallel computing, cellular automata and remote sensing.

Zi Ming is a lecturer at the School of Economics and Management, Hubei University of Technology. Her research interests include energy economy, sustainable development and spatial analysis.

Xun Liang is a professor at the School of Geography and Information Engineering, China University of Geosciences. His research interests include land use simulation and spatio-temporal modeling.

Gang Liu is a professor at the School of Compute Science, China University of Geosciences. His research interests include spatial database, spatial analysis and 3D geological modeling.

Junbin Gao is a professor at the University of Sydney Business School, the University of Sydney. His research interests include geometric deep learning, statistical learning and nonconvex optimization.

ORCID

Yang Li http://orcid.org/0000-0002-9337-1382
Mingyuan Bai http://orcid.org/0000-0002-2454-4219
Qingfeng Guan http://orcid.org/0000-0002-7392-3709
Zi Ming http://orcid.org/0000-0002-8323-0922
Xun Liang http://orcid.org/0000-0001-9401-7353
Gang Liu http://orcid.org/0000-0002-9651-4473
Junbin Gao http://orcid.org/0000-0001-9803-0256

Data and codes availability statement

The data and codes that support the findings of this study are available with the identifier at https://doi.org/10.6084/m9.figshare.23932518.

References

Azri, S., Ujang, U., and Abdul Rahman, A., 2020. Voronoi classified and clustered data constellation: a new 3D data structure for geomarketing strategies. *ISPRS Journal of Photogrammetry and Remote Sensing*, 162, 1–16.

Boyd, S.P. and Vandenberghe, L., 2014. *Convex optimization*. Cambridge, UK: Cambridge University Press.

Cheema, M.A., et al., 2011. Influence zone: efficiently processing reverse k nearest neighbors queries. In: *Proceedings of the 27th international conference on data engineering*, ICDE 2011, 11–16 April 2011, Hannover, Germany. Los Alamitos, CA: IEEE Computer Society, 577–588.

Cheema, M.A., et al., 2012. Efficiently processing snapshot and continuous reverse k nearest neighbors queries. *VLDB Journal*, 21 (5), 703–728.

Chen, F., et al., 2016. Capacity constrained maximizing bichromatic reverse nearest neighbor search. *Expert Systems with Applications*, 43, 93–108.

Fix, E. and Hodges, J.L., 1989. Discriminatory analysis. Nonparametric discrimination: consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57 (3), 238–247.

Gao, Y., et al., 2009. Visible reverse k-nearest neighbor query processing in spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, 21 (9), 1314–1327.
Gao, Y., et al., 2016. Reverse k-nearest neighbor search in the presence of obstacles. *Information Sciences*, 330, 274–292.

García-García, F., et al., 2018. Efficient large-scale distance-based join queries in spatialhadoop. *Geoinformatica*, 22 (2), 171–209.

Hjaltason, G.R. and Samet, H., 1999. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24 (2), 265–318.

Korn, F. and Muthukrishnan, S., 2000. Influence sets based on reverse nearest neighbor queries. In: *Proceedings of the 2000 ACM SIGMOD international conference on management of data*, 16–18 May 2000, Dallas, TX, USA. New York, NY: ACM, 201–212.

Li, Y. and Liu, G., 2020. Area queries based on Voronoi diagrams. In: *36th IEEE international conference on data engineering, ICDE 2020*, 20–24 April 2020, Dallas, TX, USA. Los Alamitos, CA: IEEE Computer Society, 2064–2068.

Lin, H., et al., 2016. Finding optimal region for bichromatic reverse nearest neighbor in two- and three-dimensional spaces. *Geoinformatica*, 20 (3), 351–384.

Lin, K.D., Nolen, M., and Yang, C., 2003. 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. Los Alamitos, CA: IEEE Computer Society, 290–297.

Maheshwari, A., Vahrenhold, J., and Zeh, N., 2002. On reverse nearest neighbor queries. In: *Proceedings of the 14th Canadian conference on computational geometry*, 12–14 August 2002, Alberta, Canada. Alberta, Canada: University of Lethbridge, 128–132.

Pournajaf, L., et al., 2018. Privacy preserving reverse k-nearest neighbor queries. In: *2018 19th IEEE international conference on mobile data management (MDM)*, 25–28 June 2018, Aalborg, Denmark. Los Alamitos, CA: IEEE Computer Society, 177–186.

Sharifzadeh, M. and Shahabi, C., 2010. Vor-tree: R-trees with Voronoi diagrams for efficient processing of spatial nearest neighbor queries. *Proceedings of the VLDB Endowment*, 3 (1–2), 1231–1242.

Stanoi, I., Agrawal, D., and El Abbadi, A., 2000. Reverse nearest neighbor queries for dynamic databases. In: *2000 ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, 14 May 2000, Dallas, TX, USA. New York, NY: ACM, 44–53.

Tao, Y., Papadias, D., and Lian, X., 2004. Reverse kNN search in arbitrary dimensionality. In: *Proceedings of the thirtieth international conference on very large data bases*, August 31–September 3 2004, Toronto, Canada. San Francisco, CA: Morgan Kaufmann, 744–755.

Tobler, W., 2004. On the first law of geography: a reply. *Annals of the Association of American Geographers*, 94 (2), 304–310.

Tobler, W.R., 1970. A computer movie simulating urban growth in the Detroit region. *Economic Geography*, 46 (Suppl. 1), 234–240.

Wang, S., et al., 2018. Maximize spatial influence of facility bundle considering reverse k nearest neighbors. In: *Database systems for advanced applications*, 21–24 May 2018, Gold Coast, QLD, Australia. Cham, Switzerland: Springer International Publishing, 684–700.

Wu, W., et al., 2008. FINCH: evaluating reverse k-nearest-neighbor queries on location data. *Proceedings of the VLDB Endowment*, 1 (1), 1056–1067.

Yang, S., et al., 2014. Slice: reviving regions-based pruning for reverse k nearest neighbors queries. In: *IEEE 30th international conference on data engineering, ICDE 2014*, 31 March–4 April 2014, Chicago, IL, USA. Los Alamitos, CA: IEEE Computer Society, 760–771.

Yang, S., et al., 2015. Reverse k nearest neighbors query processing: experiments and analysis. *Proceedings of the VLDB Endowment*, 8 (5), 605–616.

Yang, S., et al., 2017. Reverse k nearest neighbors queries and spatial reverse top-k queries. *VLDB Journal*, 26 (2), 151–176.

Zaheer, N., et al., 2022. Optimal school site selection in urban areas using deep neural networks. *Journal of Ambient Intelligence and Humanized Computing*, 13 (1), 313–327.