Information Systems 83 (2019) 126–144

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Moving *k*NN query processing in metric space based on influential sets

Chuanwen Li[a,*], Yu Gu[a], Jianzhong Qi[b], Rui Zhang[b], Ge Yu[a]

[a] *College of Computer Science and Engineering, Northeastern University, China*
[b] *Department of Computing and Information Systems, University of Melbourne, Australia*

HIGHLIGHTS

- Introduce the concept of the influential object in metric space.
- Use it to formulate the influential neighbor set in metric space.
- Extend the theorems previously used to the metric space.
- Propose an algorithm for moving kNN query processing in spatial networks.
- Analyze the complexity of the newly proposed algorithm.

ARTICLE INFO

Article history:
Received 7 August 2018
Received in revised form 16 February 2019
Accepted 19 March 2019
Available online 25 March 2019
Recommended by Philip Korn

Keywords:
Moving *k*NN query
k nearest neighbors
Influential set
Safe region

ABSTRACT

The moving *k* nearest neighbor query computes one's *k* nearest neighbor set and maintains it while at move. This query is gaining importance due to the prevalent use of smart mobile devices and location-based services. Safe region is a popular technique for processing the query. It is a region where the movement of the query object does not cause the query answer to change. Processing a moving *k* nearest neighbor query is a continuing process of validating the safe region and recomputing it if invalidated. The size of the safe region largely decides the recomputation frequency and hence query efficiency. Existing algorithms lack efficiency due to either computing too small safe regions frequently or computing larger but expensive safe regions. We propose to replace safe regions in metric space (e.g., Euclidean space and spatial networks) with *safe guarding objects* which have low cost to compute.We prove that, the *k* nearest neighbors stay valid as long as they are closer to the query object than the safe guarding objects are. We hence avoid the high cost of safe region recomputation. We also prove that, the region defined by the safe guarding objects is the largest possible safe region. Thus, the recomputation frequency of the proposed method is also minimized. We conduct extensive experiments comparing the proposed method with the state-of-the-art methods on both real and synthetic data. The results confirm the superiority of the proposed method.

1. Introduction

Ongoing efforts have been made to improve the user experience of mobile *location-based services* (LBS) through improving moving query processing efficiency. We revisit a major type of moving queries, the *moving k nearest neighbor* (M*k*NN) query [1–4], in metric space. *Given a moving query object and a set of static data objects, the MkNN query reports the k nearest neighbors of the query object (computed with metric distance) continuously while it is moving.* M*k*NN has many applications, such as in vehicle navigation, computer gaming, and safety engineering. For example, a Uber driver can use a M*k*NN query to track customers continuously while driving around. A computer game player can use an M*k*NN query to track nearest players whom can be attached. An underground miner can use an M*k*NN query to track nearest exits while working underground.

Computing the *k*NN of a static query object has been studied extensively (e.g., [5,6]). However, existing techniques for static *k*NN queries is not directly applicable for M*k*NN queries. This is because, if a static *k*NN query algorithm were used, keeping the *k*NN set up-to-date when the query object is moving requires constant *k*NN recomputation, which is too expensive.

Recent studies on the M*k*NN query have adapted the *safe region* based approach [2,7,8]. The idea is that objects at nearby positions often share the same *k*NN set; these nearby positions together form a region where all points have the same *k*NN set. Inside such a region, an object is "safe" to move without causing its *k*NN set to change. Thus, such a region is called a

* Corresponding author.
E-mail addresses: lichuanwen@mail.neu.edu.cn (C. Li), guyu@mail.neu.edu.cn (Y. Gu), jianzhong.qi@unimelb.edu.au (J. Qi), rui.zhang@unimelb.edu.au (R. Zhang), yuge@mail.neu.edu.cn (G. Yu).

https://doi.org/10.1016/j.is.2019.03.008
0306-4379/© 2019 Elsevier Ltd. All rights reserved.

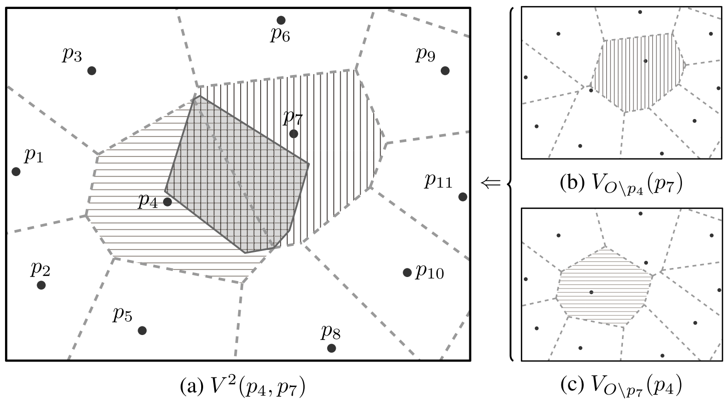


Fig. 1. Voronoi cells (order-1 and order-2).

safe region. Using the safe region significantly reduces the M*k*NN query processing cost because the *k*NN set only needs to be recomputed when the query object moves out the safe region. However, the safe region also brings in some overhead: (i) *construction overhead*: the safe region needs to be recomputed every time the *k*NN set is recomputed; (ii) *validation overhead*: whether the query object is still inside the current safe region needs to be checked every time the query location updates.

Existing methods either fall short in construction overhead or validation overhead. Earlier studies [7–9] use *Voronoi cells* as safe regions and have high construction overhead. A more recent study [2] uses relaxed safe regions to reduce construction overhead, but it suffers in frequent safe region recomputation and high validation overhead.

A Voronoi cell based safe region for *k* nearest neighbors is an *order-k Voronoi cell* [7]. Fig. 1(a) illustrates order-1 Voronoi cells on a set of objects $O = \{p_1, \ldots, p_{11}\}$, where the dashed lines denote the cell edges. Every order-1 Voronoi cell corresponds to an object inside, e.g., the top left cell corresponds to p_3. A query object inside an order-1 Voronoi cell (e.g., the cell of p_3) has the corresponding object (p_3) as its nearest neighbor. Fig. 1(a) also shows an order-2 Voronoi cell, as denoted by the cross-lined region. This cell corresponds to two objects p_4 and p_7. A query object inside this cell has $\{p_4, p_7\}$ as its 2NN set. This order-2 Voronoi cell can be computed from two order-1 Voronoi diagrams as shown in Fig. 1(b) and (c). These two order-1 Voronoi diagrams are computed without p_4 and p_7, respectively. The overlapping region of the Voronoi cells of p_4 and p_7 in these two Voronoi diagrams is the order-2 Voronoi cell of $\{p_4, p_7\}$. When *k* is a larger number, computing order-*k* Voronoi cells at query time becomes infeasible due to the high time complexity. Precomputing order-*k* Voronoi cells [8,9] is also unpractical due to the increase in the number of order-*k* Voronoi cells as *k* increases. Moreover, precomputing order-*k* Voronoi cells loses the flexibility of setting *k* at query time.

The state-of-the-art M*k*NN algorithm, V*-Diagram [2], avoids computing exact order-*k* Voronoi cells to reduce the construction overhead. It uses a safe region combined from the *safe region w.r.t. a data object* and the *fixed-rank region*, which are simpler to compute but are also less tight than the order-*k* Voronoi cells. Thus, it has to recompute the safe regions more frequently with higher validation overhead.

We propose a novel query processing approach that validates *k*NN sets as tight as order-*k* Voronoi cells, but does not have to compute exact order-*k* Voronoi cells (or any safe region at all). We use *safe guarding objects* instead of safe regions. Intuitively, since the query object moves continuously, when the *k*NN set changes, some data objects near the current *k*NN objects must become the new *k*NNs. We call this type of data objects (i.e., data objects near the current *k*NN objects) the safe guarding objects. As long as no safe guarding object becomes a *k*NN object, the current *k*NN set stays valid and hence does not need recomputation. Conceptually, the safe guarding objects define a safe region as large as the order-*k* Voronoi cell. They guarantee minimum *k*NN set recomputation and communication between the query user and the query processor. For efficient computation we identify a special set of safe guarding objects — the *influential neighbor set*. This set guarantees the validity of the *k*NN set. Meanwhile, it can be computed and validated in time linear to *k*. Therefore, it can reduce both the construction and validation overheads together. Further, this set does not require precomputation, and hence allows setting *k* at query time.

We make the following contributions:

- We analyze existing Voronoi cell based techniques for M*k*NN queries and identify possible optimizations.
- Based on the analysis, we propose the *influential set*, a novel concept that uses safe guarding objects instead of safe regions, to validate *k*NN sets in metric space.
- We propose the *influential neighbor set*, which is a special type of influential set that can be retrieved and validated efficiently.
- We propose two algorithms to process the M*k*NN query using influential neighbor sets in the Euclidean space and spatial networks, respectively. Our algorithms have the following advantages:
  - **Efficient *k*NN set validation.** We only compare the *k*NN set with a small influential neighbor set for *k*NN validation, which is simple and efficient.
  - **Efficient influential neighbor set computation.** We recompute the influential neighbor set when it becomes invalid, which is much cheaper than computing the order-*k* Voronoi cells (detailed in Section 6).
  - **Data update support.** Our algorithms support data object updates on-the-fly including insertion and deletion of data objects.
- We perform cost analysis and extensive experiments. The results confirm the superiority of our algorithms over the state-of-the-art algorithms.

This paper is an extended version of our previous paper [10]. In the previous paper, we studied the M*k*NN query in the Euclidean setting. We introduced the basic concepts of influential set and influential neighbor set, formulated an algorithm to process the M*k*NN query based on these concepts, and gave a complexity analysis on the algorithm. In this paper, we generalize the concepts to the *metric space*, i.e., a space where a *metric distance*

Table 1
Frequently used symbols.

Symbol	Meaning
k	The number of queried nearest neighbors.
ρ	The prefetch ratio. ($\rho \geq 1$)
q	The query object.
O	The set of data objects.
$d(p_i, p_j)$	The distance between objects p_i and p_j.
$b(p_i, p_j)$	The bisector between objects p_i and p_j.
$NN_k(q)$	The k nearest neighbors of q.
$MIS(O_{knn})$	The minimal influential set of a set O_{knn}.
$I(O_{knn})$	The influential neighbor set of a set O_{knn}.
$N_O(p_i)$	The Voronoi neighbor set of p_i in the Voronoi diagram with respect to object set O.
$V_O(p_i)$	The Voronoi cell of p_i in the Voronoi diagram with respect to object set O.
$V^k(O_{knn})$	The order-k Voronoi cell of object set O_{knn} in the Voronoi diagram with respect to object set O.
$A \prec_q B$	Every object in set A is nearer to q than any object in set B.
$p \curvearrowleft B$	Data object p is an influential object of set B.
$A \circlearrowleft B$	Set A is an influential set of set B.

is used as the distance measure, which satisfies non-negativity, identity of indiscernibles, symmetry, and triangular inequality. By supporting metric space, our method can apply to a larger variety of applications. For example, in a video game, MkNN queries can be used to track nearest enemy targets considering obstacles (e.g., forests, rocks, and rivers). Here, the shortest paths between a player and targets are computed under a metric space called obstructed space, where shortest paths go around obstacles instead of straight lines. Moreover, if the importance of enemy targets are considered, we can combine weighted space into the scenario, where the distances between the player and targets are divided by the importance value (weight) of each target. Obstructed and/or weighted spaces differ from the widely used Euclidean space. They are instances of metric spaces, for which our MkNN techniques can be applied. We will detail the obstructed space and the weighted space in Section 2.3. Spatial network is another widely used metric space, where the distance between two objects are defined as the length of the shortest path on spatial networks. A Uber driver should use spatial network distances to track his or her nearest customers, since Euclidean distance does not reflect the road restrictions. In this paper, we take spatial network as an example to illustrate how our method applies to the metric space. We showcase how these concepts can be applied to spatial network settings. We propose algorithms to compute and validate the influential neighbor sets in spatial networks, and analyze the algorithm complexity. We conduct further experiments to evaluate the efficiency of the proposed algorithms.

The remainder of the paper is organized as follows. The preliminaries are presented in Section 2. We propose the influential set and the influential neighbor set in Sections 3 and 4, and use them to process the MkNN query in Section 5. The cost of our algorithms is analyzed in Section 6. We report the experimental results in Section 8, review related studies in Section 9, and conclude the paper in Section 10.

2. Preliminaries

2.1. Problem definition

We first present a query definition and some basic concepts. Frequently used symbols are summarized in Table 1.

We consider a metric space where a metric distance $d(\cdot, \cdot)$ is used as the distance measure, and a query object q can move continuously in the space, i.e., as q moves from a point v to another point v', the distance between q and the two points change continuously (see Condition (iv) in Section 2.3 for a detailed definition of *continuous movement*).

A Euclidean space or a spatial network are both metric spaces. In the following discussion, we use Euclidean spaces and spatial networks to help the illustration, although the methods discussed apply to any metric space that satisfies a few simple conditions (detailed in Section 3). In such a space, the MkNN query is defined as follows.

Definition 1 (*MkNN*)**.** Given a moving query object q, a set of n static data objects $O = \{p_1, p_2, \ldots, p_n\}$, and a query parameter $k \leq n$, at every timestamp, the *moving k nearest neighbor* (MkNN) query returns a size-k subset $O_{knn} \subseteq O$ such that:

$$\forall p_i \in O_{knn}, p_j \in O \backslash O_{knn} : d(p_i, q) \leq d(p_j, q)$$

2.2. Voronoi diagram

Definition 2 (*Order-k Voronoi Diagram*)**.** The *order-k Voronoi diagram* [7] of a set of data objects O, is a subdivision of the space into *order-k Voronoi cells*, where each order-k Voronoi cell, denoted by $V^k(O_{knn})$, corresponds to a subset O_{knn} of k data objects in O. For q at any point within $V^k(O_{knn})$, O_{knn} is the kNN set of q, i.e.,

$$\forall p_i \in O_{knn}, p_j \in O \backslash O_{knn} : d(q, p_i) \leq d(q, p_j). \tag{1}$$

Fig. 1(a) gives an example in the Euclidean space, where the dots denote data objects and the dashed lines denote order-1 Voronoi cells. The cross-lined region in the middle denotes the order-2 Voronoi cell of $\{p_4, p_7\}$, $V^2(p_4, p_7)$. A query object in this cell will have $\{p_4, p_7\}$ as its 2NN set. Note that when $k > 1$, an order-k Voronoi cell does not always enclose its corresponding data objects, e.g., $V^2(p_4, p_7)$ does not enclose either p_4 or p_7.

An order-k Voronoi cell $V^k(O_{knn})$ can be computed from order-1 Voronoi cells based on the following lemma.

Lemma 1. *Given a set of data objects O and a size-k subset $O_{knn} \subseteq O$, the order-k Voronoi cell of O_{knn} is [7]:*

$$V^k(O_{knn}) = \bigcap_{p_i \in O_{knn}} V_{(O \backslash O_{knn} \cup \{p_i\})}(p_i). \tag{2}$$

Fig. 1 illustrates the computation of the order-2 Voronoi cell of $O_{knn} = \{p_4, p_7\}$. We first compute two order-1 Voronoi cells, one for p_4 (with $O \backslash O_{knn} \cup \{p_4\} = O \backslash \{p_7\}$) and the other for p_7 (with $O \backslash O_{knn} \cup \{p_7\} = O \backslash \{p_4\}$), as shown in Fig. 1(b) and (c), respectively. The intersection of the two Voronoi cells is the order-2 Voronoi cell $V^2(p_4, p_7)$.

In a spatial network, a Voronoi cell is formed by a set of network edge segments. The properties of Voronoi diagram described in Lemma 1 still hold. We consider a planar undirected (connected) graph $G = \langle V, E \rangle$ to represent a spatial network, where $V = \{v_1, v_2, \ldots, v_m\}$ represents the vertices and $E = \{e_1, e_2, \ldots, e_l\}$ represents the edges. Without loss of generality, we assume that the data objects $O = \{p_1, p_2, \ldots, p_n\}$ are all at the vertices (otherwise we can add them to the set of vertices). We use $d(o, p_i)$ to denote the shortest network distance from a point o on the graph (either at a vertex or on an edge) to a data object $p_i \in O$.

Fig. 2 shows an order-2 Voronoi diagram in a spatial network with 9 data objects $O = \{p_1, p_2, \ldots, p_9\}$. We denote the vertices with data objects by solid dots and the rest of the vertices by hollow dots, respectively. We denote the edge segments belonging to the same order-2 Voronoi cell by the same line type. For example, the three dashed edge segments connected to p_1 belong to $V^2(p_1, p_3)$, as labeled by the pair $(1, 3)$. Note that the edge segments of the same order-k ($k > 1$) Voronoi cell do not have

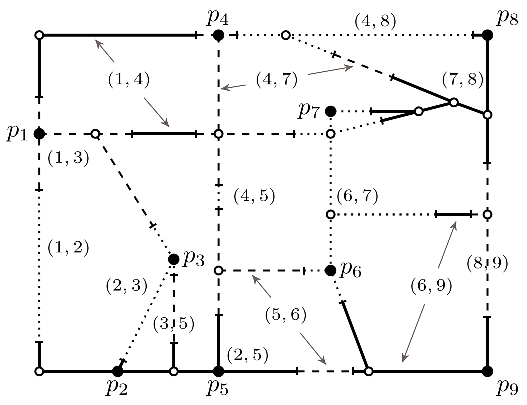


Fig. 2. Order-2 Voronoi diagram in a spatial network.

to be connected with each other, e.g., $V^2(p_1, p_4)$ consists of two separated solid edge segments at the top-left corner of the Fig.

Discussion. By definition, the order-k Voronoi cell of O_{knn} gives the largest possible safe region for O_{knn}. The kNN set of a query object is O_{knn} if and only if the query object stays in the order-k Voronoi cell of O_{knn}. Ideally, the MkNN query can be processed by precomputing the order-k Voronoi diagram, where the kNN sets are reported as the query object moves into different order-k Voronoi cells. However, as Lemma 1 suggests, computing an order-k Voronoi cell requires computing lower order Voronoi cells. *This computation is expensive especially when k is a large number. It is also inflexible with regard to changes of k or the data set O, which is a significant disadvantage [7].*

2.3. Generalizing the Voronoi diagram

We proceed to generalize the Voronoi diagram to metric space. Given a non-empty space V and a set of data objects $O = \{p_1, \ldots, p_n\}$ in the space, we consider an assignment δ which maps a point $v \in V$ to elements of O:

$$\delta(v, p_i) = \begin{cases} 1 & \text{if } v \text{ is assigned to } p_i \text{ in } O, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

We denote the set of points assigned to p_i as $V_O(p_i)$:

$$V_O(p_i) = \{v \mid \delta(v, p_i) = 1,\ \ v \in V\}. \tag{4}$$

Further, we denote the set of points assigned to both p_i and p_j as $b(p_i, p_j)$:

$$b(p_i, p_j) = V_O(p_i) \cap V_O(p_j),\ \ i \neq j. \tag{5}$$

Okabe et al. [7] identified the following three conditions on the assignment δ to guarantee the existence of a Voronoi diagram fashioned tessellation in the metric space V.

- **Condition 1.** Every point $v \in V$ is assigned to at least one element of O:
$$\sum_{i=1}^{n} \delta(v, p_i) \geq 1, v \in V. \tag{6}$$
- **Condition 2.** There exists at least two sets $V_O(p_i)$ and $V_O(p_j)$ which have positive Lebesgue measure, e.g., the *areas* of $V_O(p_i)$ and $V_O(p_j)$ in a 2D space, or the *volumes* of $V_O(p_i)$ and $V_O(p_j)$ in a 3D space, are non-zero.
- **Condition 3.** If $b(p_i, p_j)$ is not empty, $b(p_i, p_j)$ should coincide with the intersection of the boundaries of $V_O(p_i)$ and $V_O(p_j)$:
$$b(p_i, p_j) = \partial V_O(p_i) \cap \partial V_O(p_j),\ \ i \neq j. \tag{7}$$

Here, $\partial V_O(p_i)$ denotes the boundary of $V_O(p_i)$.

Condition 3 implies that $V_O(p_i)$ and $V_O(p_j)$ are mutually exclusive except for their boundaries. This condition guarantees that $V_O(p_1), \ldots, V_O(p_n)$ are collectively exhaustive.

An assignment that satisfies these three conditions will generate a set of point sets $V_O(p_1), \ldots, V_O(p_n)$ that are collectively exhaustive and mutually exclusive except for their boundaries. These point sets form a tessellation of V, which is called the *generalized Voronoi diagram* generated by the set O with an assignment δ in V; these point sets themselves are called the *generalized Voronoi cells* [7].

Our focus is on the nearest neighbors, and hence we use the distance function to form an assignment δ:

$$\delta(v, p_i) = \begin{cases} 1 & d(v, p_i) \leq d(v, p_j), \quad \forall i, j \in \{1, \ldots, n\}, \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

Here, $d(v, p_i)$ denotes the distance between a point v and an object p_i. It is defined based on some *distance metric* in the space V. We require the distance between a point and an object to satisfy the following four axioms: given an object p_i in O and points v, v' in V,

- $d(v, p_i) = 0 \iff v = p_i$;
- $d(v, p_i) \leq d(v', p_i) + d(v, v')$;
- $d(v, p_i) = d(p_i, v)$;
- $v \neq p_i \implies d(v, p_i) > 0$.

For continuous queries we need the following condition to guarantee the continuity of the object movement.

- **Condition 4.** For a query point moving continuously from point v to v' in V, the distance from the query point to any object $p_i \in O$ changes continuously, i.e.,
$$\lim_{d(v,v') \to 0} d(v, p_i) - d(v', p_i) = 0,\ \forall v, v' \in V,\ \forall p_i \in O. \tag{9}$$

The Voronoi diagram discussed in Section 3 is a special case of the generalized Voronoi diagram in the Euclidean space where $d()$ is the Euclidean distance function. It satisfies all four conditions above straightforwardly.

There are many other problems that can also be formalized as generalized Voronoi diagrams [7], and hence our MkNN algorithm may apply. Note that we do not need the problem to be in a metric space. We only need that, for supporting Condition 4, the distance between a point and an object is a metric distance. We call a space *generalized* metric space if the distance between any point to any object is a metric distance, so that the generalized Voronoi diagrams can be defined such spaces. A metric space is a generalized metric space, but the opposite is not true. The difference is that, in a generalized metric space, the distances between two arbitrary points may not have been defined. By supporting generalized metric spaces, our method applies in more scenarios, e.g., weighted Euclidean space as illustrated in Fig. 3(b).

Fig. 3 shows two examples. In Fig. 3(a), a few obstacles (denoted by the gray regions) are added to the space. The distance function is defined as the length of the shortest path between two points bypassing the obstacles. Then, the regions separated by the gray line segments show a tessellation on the space for the set of objects $\{p_1, p_2, \ldots, p_{13}\}$. We call this kind of spaces obstructed spaces. In an obstructed space, a static kNN query is processed as follows. We first compute a *visibility network*. All objects and obstacle vertices are used as vertices of the visibility network. We add an edge between an object and an obstacle vertex if there is no other obstacles lying in between (i.e., being "visible"). When a query comes, we connect it to the visibility network by adding the query point into the network as a vertex and connecting it

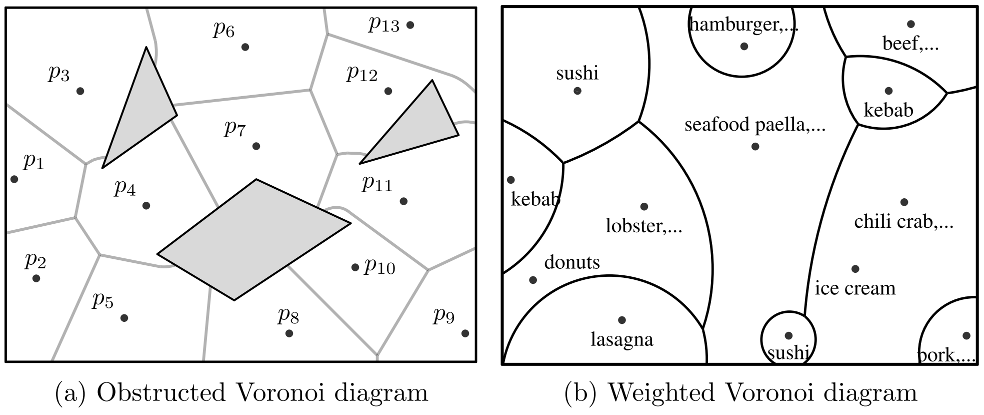


Fig. 3. Generalized Voronoi diagrams.

(i.e., adding edges) to its visible vertices. Then, we run Dijkstra's algorithm from the query point on the visibility network to find its *k*NN. Please see our previous paper for detail [11].

Fig. 3(b) shows an example of the weighted Euclidean space, a generalized metric space where generalized Voronoi diagrams can be used for processing spatial keyword queries. In a weighted Euclidean space, each data object is represented by a dot (e.g., a restaurant), and each dot is associated with a weight. Here, the weight of an object can be computed as the similarity between the set of its attributed keywords (e.g., cuisine served by the restaurant such as sushi) and the queried keyword. Then, the weighted distance between a query and an object is defined as the Euclidean distance divided by the weight of the data object.

Formally, given an object p_i with weight w_i, the weighted distance from a point v to p_i is defined as $d(v, p_i)/w_i$. Note that the weighted distance between two points or two objects do not need to be defined. This is because we do not require the weighted space to be a metric space. We only need that, when a query point v moving towards to an object p_i (Condition 4), the distance between them can be computed. Here, the weighted Euclidean space satisfies our generalized metric space requirement.

A static *k*NN query in a weighted Euclidean space can be processed in a way similar to that in the Euclidean space, except that the weighted distance is considered instead of the Euclidean distance. See Section 5.4 for a detailed discussion on weighted distance.

2.4. V*-Diagram

V*-Diagram is the state-of-the-art solution for M*k*NN queries in both Euclidean spaces and spatial networks [2,12]. We briefly describe the algorithm below and will use it as the baseline algorithm in the experiments. V*-Diagram avoids the high cost of order-*k* Voronoi cell computation by using an *Integrated Safe Region* (ISR). The ISR is the intersection of two types of regions: (i) the *safe region w.r.t. the kth NN*, and (ii) the *fixed-rank region* of the $k + x$ NNs, where x is the number of *auxiliary data objects* maintained to reduce safe region recomputation.

V*-diagram in Euclidean space. We use Fig. 4(a) to illustrate V*-Diagram, where $k = 2$ and $x = 2$. Let q_c be the current position of the query object q. The algorithm starts with computing the query object's $(k+x)$ nearest data objects, which are p_1, p_2, p_3, p_6 in the figure. Let z be the current $(k+x)^{th}$ nearest data object (p_6). The goal is to find the locations where q can move to from q_c without causing the *k*NN set to change. These locations together form the safe region. First, a *known region* is computed as a disk centered at q_c with the radius $d(q_c, z)$, which is denoted by the solid line circle. This region encloses the $(k + x)$ nearest data objects. Next, for each of the $(k + x)$ nearest data objects p_i, a safe region is computed to guarantee that p_i is closer to q than any object outside the know region.

Step 1. The *safe region w.r.t. a data object* p_i, $\omega(q_c, p_i, d(q_c, z))$, is computed as a region that, when q is at any point q' in the region, p_i is nearer to q than any data object p_j outside the known region. Formally,

$$\omega(q_c, p_i, d(q_c, z)) = \{q' | d(q', p_i) + d(q_c, q') \le d(q_c, z)\}.$$

Nutanong et al. [2] show that the inequality above defines an ellipse where q_c and p_i are the two focal points and $d(q_c, z)$ is the major axis length. In Fig. 4(a), the vertical-lined region denotes $\omega(q_c,\ p_3,\ d(q_c, p_6))$, where q_c and p_3 are the two focal points and q moving in this region guarantees that p_3 is closer to q than any data object outside the known region. Similarly, the horizontal-lined region is the safe region w.r.t. p_1, i.e., $\omega(q_c,\ p_1,\ d(q_c, p_6))$.

As long as q is inside the intersection of the safe regions w.r.t. the top k data objects in the known region, i.e., $\bigcap_{i=1}^{k} \omega(q_c, p^i, d(q_c, z))$, it is guaranteed that any data object p_j outside the known region cannot be closer to q than any of those k data objects. Here, p^i denotes the *i*th NN. In Fig. 4(a), the cross-lined region denotes such a region.

Step 2. V*-Diagram further computes a region called the *fixed-rank region* where the order of nearness of the k+x data objects to q does not change, either. Formally, the fixed-rank region of a list L_{k+x} of k+x ranked data objects, $\eta\langle p^1, p^2, \ldots, p^{k+x}\rangle$, is defined as the intersection of the dominant region of p^i and p^{i+1}, $H(p^i, p^{i+1})$, where p^i is nearer to q than p^{i+1} ($i \in [1 \cdots k + x - 1]$):

$$\eta\langle p^1, p^2, \ldots, p^{k+x}\rangle = \bigcap_{i=1}^{k+x-1} H(p^i, p^{i+1}).$$

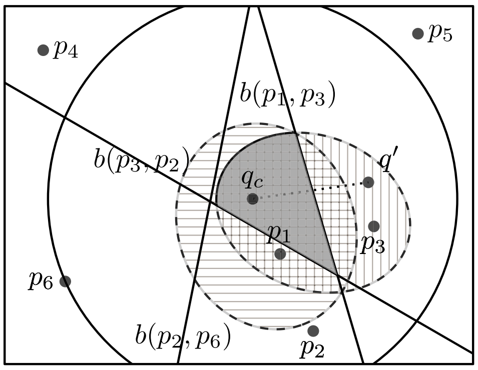
Step 3. The intersection of the safe regions w.r.t. the k data objects and the fixed-rank region is the *Integrated Safe Region (ISR)*, denoted by $\Omega(q_c, L_{k+x})$. Formally,

$$\Omega(q_c, L_{k+x}) = \eta(L_{k+x}) \cap (\bigcap_{i=1}^{k} \omega(q_c, p^i, d(q_c, z))),$$

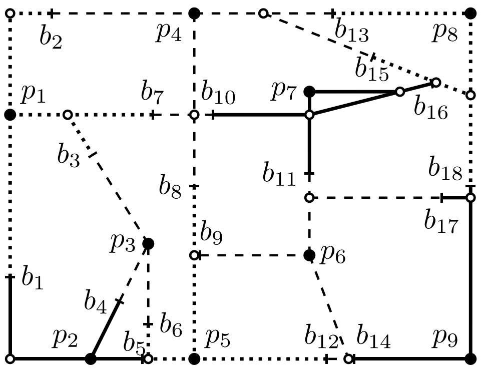
where p_k denotes the *k*th nearest data object of q. This computation can be simplified as follows [2]:

$$\Omega(q_c, L_{k+x}) = \eta(L_{k+x}) \cap \omega(q_c, p^k, d(q_c, z)).$$

V*-diagram in spatial networks. In spatial networks, similar concepts apply, although now the distance between two points becomes their shortest path distance and the boundary of a safe region becomes a set of discrete boundary points. Nutanong et al. [12] use a graph breadth-first search starting from the query



(a) ISR ($k = 2$, $x = 2$)



(b) A network Voronoi diagram

Fig. 4. Examples of V*-Diagram and VN^3.

object location to find the boundary points of the safe regions w.r.t. data objects and the fixed-rank regions.

Discussion. V*-Diagram uses the safe region w.r.t. the kth NN to replace the Voronoi cells, which reduces the safe region computation complexity. However, the safe region obtained is not as large as the order-k Voronoi cell. The safe regions will become invalid more frequently, meaning that the kNN set will be recomputed more frequently, and will bring in unnecessary query processing overhead.

2.5. The VN^3 algorithm

The *Voronoi-based Network Nearest Neighbor (VN^3)* [13] algorithm is an efficient algorithm for static kNN queries in spatial networks. We adapt it to process MkNN queries in spatial networks. The algorithm works as follows.

Step 1. *Indexing the Voronoi diagram.* VN^3 first converts the network into an order-1 network Voronoi diagram, where each data object p_i corresponds to a network Voronoi cell. The Voronoi cells are bounded by *Minimum Bounding Rectangles (MBR)* and indexed by an R*-tree [14].

Step 2. *Precomputing the distance between the Voronoi cell boundary points and the data objects.* Neighboring Voronoi cells share boundary points. These boundary points can be seen as the "gates" of the cells, since q must pass a boundary point when its NN changes. For efficient on-line NN computation, the distance between a data object and all of its Voronoi cell boundary points, plus the distance between the boundary points themselves, are precomputed.

Fig. 4(b) shows the precomputed Voronoi diagram of $O = \{p_1, p_2, \dots, p_9\}$, where edge segments that belong to the same Voronoi cell are denoted by the same line type. The boundary points of the cells are denoted by $\{b_1, b_2, \dots, b_{18}\}$, e.g., the boundary points of $V(p_1)$ are b_1, b_2, b_3, and b_7. The distance between p_1 and these points, $d(p_1, b_1)$, $d(p_1, b_2)$, $d(p_1, b_3)$, and $d(p_1, b_7)$, as well as the distance between these boundary points $d(b_1, b_2)$, $d(b_1, b_3)$, $d(b_1, b_7)$, $d(b_2, b_3)$, $d(b_2, b_7)$, and $d(b_3, b_7)$, are precomputed.

It has been proven [13] that the $(k + 1)$st NN of q must be a neighboring object (i.e., a *Voronoi neighbor*, defined in Section 3) of at least one of its top kNNs, and that the path from q to its $(k+1)$st NN only passes through the Voronoi cells of the top kNNs.

Step 3. *Query processing.* Once the network Voronoi diagram and the shortest path distances have been computed, a kNN query is processed as follows. As summarized in Algorithm 1, first, the Voronoi cell that contains q is located, which is done by a function called Contain(q) performing a point query on the R*-tree. The data object corresponding to the Voronoi cell is added to the result set O_{knn} and its Voronoi neighbors are added to a set denoted by N (Line 1). Then, while there are fewer than k objects in O_{knn}, VN^3 repeatedly adds an object $p_i \in N$ that is the nearest to q into O_{knn}, and adds its Voronoi neighbors to N (Lines 2 to 6). When the loop terminates, O_{knn} is the result set (Line 7). Note that in the algorithm, the function ShortestDistance (q, p_i) computes the shortest path distance between q and p_i using the precomputed shortest path distances between the Voronoi cell boundary points.

Algorithm 1 Finding kNN Set

```
Input: a query object $q$
Output: the $k$NN set of $q$
1: $O_{knn} \leftarrow$ Contain($q$),
2: $N \leftarrow$ Contain($q$).neighbors
3: while $|O_{knn}| < k$ do
4:     $p_i \leftarrow \arg\min_{p_i \in N}$ ShortestDistance($q, p_i$)
5:     $O_{knn} \leftarrow O_{knn} \cup \{p_i\}$
6:     $N \leftarrow N \cup p_i.neighbors \setminus O_{knn}$
7: return $O_{knn}$
```

Discussion. Applying VN^3 to process a MkNN query straightforwardly requires re-running the algorithm every time the query object changes its location, which may incur prohibitive cost. In Section 5 we will show how to adapt the algorithm to avoid the high re-running costs.

3. Influential set

Our key idea is to use *safe guarding objects* to replace safe regions. As long as the query object is closer to the kNNs than to the safe guarding objects, it is guaranteed that the kNNs are valid, and no recomputation is needed. We call a set of safe guarding objects an *influential set*, for that they are *influential* in deciding the validity of a kNN set.

Definition 3 (*Influential Set, IS*)**.** Given a set of data objects O, a query object q and a kNN set $O_{knn} \subseteq O$, we call a set $S \subseteq O \backslash O_{knn}$ an *influential set* of O_{knn}, denoted by $S \circlearrowleft O_{knn}$, if O_{knn} stays as the kNN set of q as long as objects in O_{knn} are closer to q than objects in S are, i.e.,

$$O_{knn} = NN_k(q) \iff O_{knn} \prec_q S. \tag{10}$$

Here, $NN_k(q)$ is a function that returns the kNN set of q, and $A \prec_q B$ denotes that any object in a set A is closer to q than every object in a set B.

Trivially, there always exists an IS for every kNN set O_{knn}, i.e., $O \backslash O_{knn}$, since by definition the kNN set O_{knn} is closer to q than any object in $O \backslash O_{knn}$. However, $O \backslash O_{knn}$ is too large for validation purpose.

We proceed to find a smaller IS. We start by defining the *influential object* (IO), and prove that every edge of $V^k(O_{knn})$ is contributed by the bisector between an object in O_{knn} and an influential object. Our definition of the influential object is based on the Voronoi diagram. In a metric space, there are four conditions required so that a (generalized) Voronoi diagram exists [7], which is detailed in Section 2.3. Essentially, they require that every point in the space can be uniquely assigned to a nearest object in the space.

Lemma 2. *Any edge of $V^k(O_{knn})$ is a segment of the bisector between two objects p_i and p_j, where:*

$$p_i \in O_{knn};\ V^{k+1}(O_{knn} \cup \{p_j\}) \backslash V^k(O_{knn}) \neq \emptyset. \tag{11}$$

We call object p_j an influential object *(IO) of O_{knn}, denoted by $p_j \curvearrowleft O_{knn}$, if and only if it satisfies the condition above.*

Proof. By definition, any point v on an edge of $V^k(O_{knn})$ must be on the bisector between some object $p_i \in O_{knn}$ and some object $p_j \notin O_{knn}$. When the query object q is at v, p_j and p_i have the same distance to q. Meanwhile, p_j and p_i must be nearer to q than objects in $O \backslash (O_{knn} \cup \{p_j\})$ are, otherwise p_i must not be a kNN object. Similarly, p_j and p_i must be farther to q than objects in $O_{knn} \backslash \{p_i\}$ are, otherwise p_j must be a kNN object.

Since q moves continuously, the distance between q and any data object changes continuously. Let q move slightly from v to a point v' just outside $V^k(O_{knn})$. Then the distance between p_i and q is slightly greater than that between p_j and q. If v and v' are close enough, p_i and p_j are still nearer to q than objects in $O \backslash (O_{knn} \cup \{p_j\})$ and farther to q than objects in $O_{knn} \backslash \{p_i\}$. Now p_i has become the $(k+1)$st NN, p_j the kth NN, and $O_{knn} \backslash \{p_i\}$ the $k-1$ NNs. Thus, we have a point v' belonging to $V^{k+1}(O_{knn} \cup \{p_j\})$ but not $V^k(O_{knn})$, which means $V^{k+1}(O_{knn} \cup \{p_j\}) \backslash V^k(O_{knn}) \neq \emptyset$.

Fig. 5 illustrates the lemma. Fig. 5(a) is an order-3 Voronoi diagram in an Euclidean space, where $k = 2$ and the current 2NN set is $\{p_4, p_7\}$. The gray region in the middle is $V^2(p_4, p_7)$. The influential objects are those whose corresponding order-3 Voronoi cells overlap with $V^2(p_4, p_7)$. We see that $V^3(p_3, p_4, p_7)$, $V^3(p_4, p_5, p_7)$, $V^3(p_4, p_6, p_7)$, $V^3(p_4, p_7, p_8)$, and $V^3(p_4, p_{10}, p_7)$ overlap with $V^2(p_4, p_7)$. Thus, the corresponding data objects p_3, p_5, p_6, p_8, and p_{10} are the influential objects. Each of them contributes at least an edge of $V^2(p_4, p_7)$, e.g., the edges in $V^3(p_4, p_7, p_8)$ is contributed by $b(p_4, p_8)$.

Fig. 5(b) is an example in a spatial network. The thick gray line segments denote $V^2(p_6, p_7)$. The order-3 Voronoi cells $V^3(p_4, p_6, p_7)$, $V^3(p_5, p_6, p_7)$, $V^3(p_6, p_7, p_8)$, and $V^3(p_6, p_7, p_9)$ overlap with $V^2(p_6, p_7)$. Thus, the objects p_4, p_5, p_8, and p_9 are the influential objects, which contribute edges of $V^2(p_6, p_7)$, e.g. the boundary point of $V^2(p_6, p_7)$ in $V^3(p_6, p_7, p_9)$ is contributed by $b(p_6, p_9)$ and $b(p_7, p_9)$.

Fig. 5(c) shows an example in a less common metric space – a *weighted Euclidean space*. In this space, every object p is associated with a real value as its weight, denoted by $p.w$. The distance of two objects p_i and p_j, $d(p_i, p_j)$, is defined as their Euclidean distance multiplied by their weights, i.e., $d(p_i, p_j) = p_i.w \cdot p_j.w \cdot d_e(p_i, p_j)$, where $d_e(p_i, p_j)$ denotes the Euclidean distance. In the figure, the number inside parenthesis next to each object represents the object's weight, e.g., the weight of p_7 is 7. The gray region is the weighted Voronoi cell of p_7, $V_O^1(p_7)$. Its each edge is dashed and labeled with the pair of objects forming the edge, e.g., (6, 7) refers to the edge between p_6 and p_7. The order-2 weighted Voronoi cells (bounded by the solid lines) $V_O^2(\{p_3, p_7\})$, $V_O^2(\{p_6, p_7\})$, $V_O^2(\{p_7, p_{13}\})$, $V_O^2(\{p_7, p_{12}\})$, $V_O^2(\{p_7, p_{11}\})$, $V_O^2(\{p_7, p_8\})$, $V_O^2(\{p_5, p_7\})$, and $V_O^2(\{p_4, p_7\})$ overlap with $V_O^1(p_7)$. Thus, p_3, p_4, p_5, p_6, p_8, p_{11}, p_{12}, and p_{13} are the influential objects.

We now prove that an object set S is an influential set if and only if S contains all the influential objects of O_{knn}.

Theorem 1. *Given a kNN set O_{knn} and a set $S \subseteq O \backslash O_{knn}$,*

$$S \circlearrowleft O_{knn} \iff S \supseteq \bigcup_{p_i \curvearrowleft O_{knn}} \{p_i\}. \tag{12}$$

We call $\bigcup_{p_i \curvearrowleft O_{knn}} \{p_i\}$ the minimal influential set *(MIS) of O_{knn}, denoted by $MIS(O_{knn})$.*

Proof. (i) $S \circlearrowleft O_{knn} \Rightarrow S \supseteq MIS(O_{knn})$: We prove by contradiction. Suppose $S \not\supseteq MIS(O_{knn})$. Then there must be an object $p_i \in MIS(O_{knn})$ and $p_i \notin S$. According to Lemma 2, $V^{k+1}(O_{knn} \cup \{p_i\}) \backslash V^k(O_{knn})$ is not empty. Let the query object q be in $V^{k+1}(O_{knn} \cup \{p_i\}) \backslash V^k(O_{knn})$. Now $NN_{k+1}(q) = O_{knn} \cup \{p_i\}$ but $NN_k(q) \neq O_{knn}$, which means that $O_{knn} \prec_q S$ but O_{knn} is not the kNN set. Hence $S \circlearrowleft O_{knn}$ does not hold.

(ii) $S \supseteq MIS(O_{knn}) \Rightarrow S \circlearrowleft O_{knn}$: Given $S \supseteq MIS(O_{knn})$: (a) if $O_{knn} = NN_k(q)$, then by the definition of the kNN query, we have $O_{knn} \prec_q S$; (b) if $O_{knn} \prec_q S$, since $S \supseteq MIS(O_{knn})$, Lemma 2 guarantees that the query object q must be enclosed by $V^k(O_{knn})$, and hence $O_{knn} = NN_k(q)$.

Discussion. Similar to computing a strict safe region (an order-k Voronoi cell), computing the MIS is an expensive operation.[1] However, by Definition 3 and Theorem 1, *a larger IS does not affect the strictness in validating the* kNN *sets (i.e., it still defines the exact order-k Voronoi cell).* This is an important advantage of our IS based method over safe region based methods as an approximated safe region must be smaller than the order-k Voronoi cell since it should be enclosed by the cell. Therefore, to reduce the computational cost, we can compute instead an IS that is slightly larger than the MIS but with much higher computational efficiency.

Instead of the MIS, various kinds of IS may be used for kNN validation, which may bring a series of new studies on the MkNN query. In next section we propose a kind of IS called the *influential neighbor set*.

4. Influential neighbor set

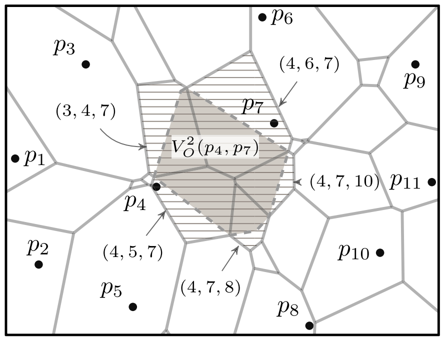
The *influential neighbor set* is based on *Voronoi neighbors*.

Definition 4 (*Voronoi Neighbor*)**.** Given an object set O, two objects $p_i, p_j \in O$ are Voronoi neighbors if and only if there exists a point b where $d(b, p_i) = d(b, p_j)$, and there is no other object $p \in O$ satisfying $d(b, p) < d(b, p_i)$[7]. We denote the set of all Voronoi neighbors of p by $N_O(p)$.

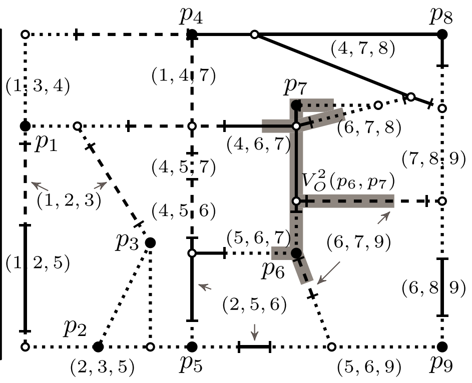
Intuitively, if two objects are Voronoi neighbors, their order-1 Voronoi cells share an edge. Since order-1 Voronoi diagrams can be precomputed and stored with little overhead [16], Voronoi neighbors can be computed efficiently.

We use Voronoi neighbors to approximate the minimal influential set of a kNN set O_{knn} ($k > 1$). We prove that the union of the Voronoi neighbors of each object in O_{knn} forms a good approximation of $MIS(O_{knn})$. For ease of discussion we call this union set the *influential neighbor set* (INS).

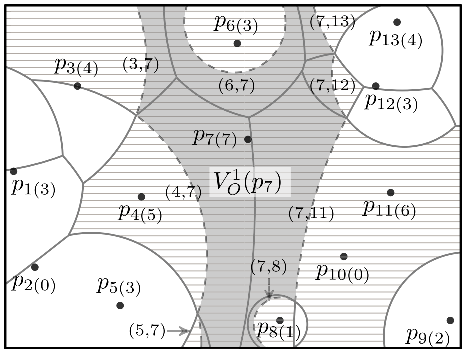
[1] In Euclidean space, an order-k Voronoi diagram on n points has $\mathcal{O}(k(n-k))$ cells [7] and computes in $\mathcal{O}(k(n-k)\log n + n\log^3 n)$ time [15].



(a) In Euclidean space



(b) In spatial network



(c) In weighted Euclidean space

Fig. 5. Influential objects.

Definition 5 (*Influential Neighbor Set, INS*)**.** Given an object set O and a kNN set $O_{knn} \subset O$, we call the union of all the Voronoi neighbors of $p_i \in O_{knn}$, excluding those already in O_{knn}, the *influential neighbor set* (INS) of O_{knn}, denoted by $I(O_{knn})$. Formally,

$$I(O_{knn}) = (\bigcup_{p_i \in O_{knn}} N_O(p_i)) \backslash O_{knn}. \tag{13}$$

Next we prove $I(O_{knn}) \supseteq MIS(O_{knn})$, which means that an INS is an IS, based on the following lemma.

Lemma 3. *Given an object set O, let p_i be an object in O, o be any point in the space, and $d(o, p_i)$ be their distance. Let N be the subset of O containing all the objects whose distances to o are smaller than or equal to $d(o, p_i)$, i.e.,*

$$N = \{p_j \in O\backslash\{p_i\} | d(o, p_j) \le d(o, p_i)\}.$$

If $N \neq \emptyset$, then there exists an object $p \in N$ such that p_i and p are Voronoi neighbors.

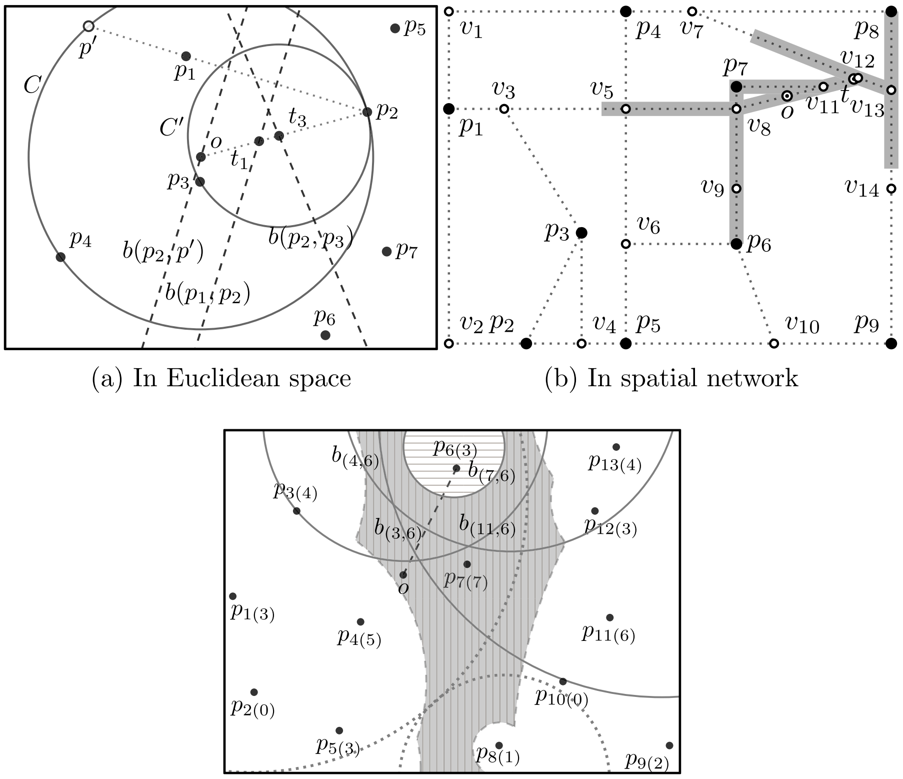
Proof. We call a point t_j on the shortest path from o to p_i the *intersection point* of $p_j \in N$ if $d(t_j, p_j) = d(t_j, p_i)$. For each object $p_j \in N$, there exists at least one intersection point. This is due to the continuity of the object movement in our problem assumption. When an object q moves from o to p_i, the distance from q to p_i decreases continuously from $d(o, p_i)$ to 0. Meanwhile, by definition $d(o, p_j) \le d(o, p_i)$. At some point, $d(q, p_i)$ will become equivalent to $d(q, p_j)$. This point is the intersection point t_j.

Among all the intersection points of the objects in N, let the one nearest to p_i be t and its corresponding object be p. There is no object in N that is nearer to t than p is, otherwise the object's intersection point is nearer to p_i than t. There is no object in $O\backslash N$ that is nearer to t than p either, otherwise it is nearer to o than p_i is and should belong to N. Therefore, by Definition 4, p_i and p are Voronoi neighbors.

Fig. 6(a) illustrates the lemma in an Euclidean space. For a point o and an object p_2, the set $N = \{p_1, p_3, p_4\}$ contains objects enclosed by the circle C (the larger circle) that centers at o and crosses p_2 on its boundary. For each object p_j in N, the bisector $b(p_j, p_2)$ must intersect the line segment between o and p_2, $\overline{o, p_2}$ (i.e., the shortest path between the two points) at some point denoted by t_j. This is because, if p_j were a point p' on the boundary of C, then by the property of a circle the bisector $b(p_2, p')$ must pass through the center o of C. If p_j is enclosed by C rather than on C, the bisector $b(p_2, p_j)$ must be closer to p_2 than $b(p_2, p')$ is and hence must intersect $\overline{o, p_2}$. For p_3 whose intersection point t_3 is the nearest to p_2 among all the intersection points, the circle C' centered at t_3 with $d(p_2, t_3)$ as the radius (the smaller circle) contains no other object inside. Thus, by Definition 4, p_3 and p_2 are Voronoi neighbors.

Fig. 6(b) illustrates the lemma in a spatial network, where the gray line segments denote the area nearer to o than p_8. The objects in this area, p_6 and p_7, constitute N. On the shortest path from o to p_8, for each object $p_j \in N$, we locate the first network vertex v that is nearer to p_j than to p_8, e.g., for p_7, v_{11} is the first vertex on the shortest path from o to p_8 that is nearer to p_7 than to p_8. If there is no such vertex (i.e., all the vertices on the path are nearer to p_8), we treat o as the vertex v, which is the case for p_6. Then the intersection point t_j between p_8 and p_j is located on the shortest path from v to p_j with a distance of $\frac{d(v,p_j)-d(v,p_8)}{2}$ from v. In the figure, t denotes the intersection point that is nearer to p_8. It is the intersection point between p_7 and p_8, while o itself is the intersection point between p_6 and p_8. Thus, p_7 and p_8 are Voronoi neighbors.

Fig. 6(c) illustrates the lemma in a weighted Euclidean space. We denote the bisectors between p_6 and other objects by solid or



(a) In Euclidean space (b) In spatial network

(c) In weighted Euclidean space

Fig. 6. Examples of Lemma 3.

dotted arcs/lines. The shortest path from o to p_6 crosses the solid arcs/lines, which are $b(p_{11}, p_6)$, $b(p_3, p_6)$, $b(p_4, p_6)$, and $b(p_7, p_6)$. This indicates that p_{11}, p_3, p_4, and p_7 are nearer to o than p_6 is. By the lemma, there should be a Voronoi neighbor of p_6 among them. In the figure, we can see that p_7 is the Voronoi neighbor, since $V_O(p_7)$ (the gray area) and $V_O(p_6)$ (the horizontal lined area) are adjacent.

Theorem 2. *Given an object set O, a* kNN *set O_{knn}, and the corresponding $I(O_{knn})$ as defined in Definition 5:*

$$I(O_{knn}) \circlearrowleft O_{knn}. \tag{14}$$

Proof. We prove by showing that $I(O_{knn}) \supseteq MIS(O_{knn})$, i.e., any object $p \in MIS(O_{knn})$ is included in $I(O_{knn})$. We show that for any object $p \in MIS(O_{knn})$, there exists an object $p_i \in O_{knn}$ that is a Voronoi neighbor of p.

Since $p \in MIS(O_{knn})$, $V^{k+1}(O_{knn} \cup \{p\}) \backslash V^k(O_{knn}) \neq \emptyset$ by definition. For any point q in $V^{k+1}(O_{knn} \cup \{p\}) \backslash V^k(O_{knn})$, it views p as one of its kNNs. It is nearer to p than to some data object $p^k \in O_{knn}$, i.e., $NN_k(q) = O_{knn} \backslash \{p^k\} \cup \{p\}$. Meanwhile, by definition, for any point q' in $V^k(O_{knn})$, it is nearer to p^k than to p. As we assume continuous movement of the query object, the distance between the objects changes continuously. If the query object moves inside $V^{k+1}(O_{knn} \cup \{p\})$ (along any path) from q' to q, there must be a point o on the path where the kNN set changes from O_{knn} to $O_{knn} \backslash \{p^k\} \cup \{p\}$. Both p and p^k have the same distance to o. Since o is inside $V^{k+1}(O_{knn} \cup \{p\})$, the other objects in $O_{knn} \cup \{p\}$ must be nearer to o than p (or p^k). By Lemma 3 we know that there must be an object in O_{knn} that is a Voronoi neighbor of p. Thus, $p \in I(O_{knn})$ and the theorem is proven.

Algorithm 2 QueryMaintenance

```
Input: query object $q$, prefetched set $R$, current $k$NN set $O_{knn}$ and
    its influential set $I$
Output: $k$NN set at each timestamp
1: while true do
2:     $r \leftarrow$ Validate($q$, $O_{knn}$, $I$)
3:     if $r.isValid$ = false then
4:         Update($q$, $R$, $O_{knn}$, $I$, $r.candidate$, $r.delete$)
5:     Process updates of $O$
```

5. Query processing

A MkNN query is processed as follows. We compute an initial $\lfloor \rho k \rfloor$ NN set ($\rho \geq 1$), denoted by R, and its INS, $I(R)$. We call ρ the *prefetch ratio*, which is a system parameter. The extra $\lfloor (\rho - 1)k \rfloor$ NNs play a similar role to that of the x extra NNs in the V*-Diagram algorithm [2]. They balance the query result communication and recomputation costs. Intuitively, a larger ρ yields a large safe region (conceptually) and hence lower communication frequency between the query processor and the query user; but it also takes a longer time to compute a larger $\lfloor \rho k \rfloor$ NN set. We evaluate the impact of ρ and choose the optimal value empirically.

For fast INS computation, we precompute and index the order-1 Voronoi diagram of O (using an VoR-tree [16] in the Euclidean space and the VN^3 algorithm [13] in a spatial network). Since we do not need to precompute order-k Voronoi cells, our algorithm can answer kNN queries where k is determined at query time.

We return the sets R and $I(R)$, where the top k objects of R form the kNN set O_{knn} and $I(R) \cup R \backslash O_{knn}$ form the IS. Then query maintenance starts. At each timestamp, we check whether

Algorithm 3 Validation

```
Input: a query object $q$
Output: whether $O_{knn}$ is the $k$NN of $q$
 1: if Contain($q$) $\notin O_{knn}$ then
 2:     return false
 3: $O'_{knn}$ ←Contain($q$),
 4: $N \leftarrow$ Contain($q$).neighbors
 5: while $|O'_{knn}| < k$ do
 6:     find $p_i$ with minimum ShortestDistance($q, p_i$) from $N$
 7:     if $p_i \notin O_{knn}$ then
 8:         return false
 9:     else
10:         $O'_{knn} \leftarrow O'_{knn} \cup \{p_i\}$
11:         $N \leftarrow N \cup p_i.neighbors \setminus O'_{knn} \setminus \{p_i\}$
12: return true
```

the current kNN set is nearer to q than the IS: (i) if yes, the current kNN set is still valid, as guaranteed by Theorem 2; (ii) if not, we update the kNN set. Algorithm 2 summarizes the query maintenance process.

5.1. kNN set validation

In Algorithm 2, we use the validation function to test whether the current kNN set is nearer to q than the IS is.

In a general metric space, the validation function finds the object in O_{knn} that is the farthest from q, denoted by $r.delete$, and the object in the IS that is the nearest to q, denoted by $r.candidate$. If $r.candidate$ is closer to q than $r.delete$, the kNN set has become invalid and we use the function Update to update the kNN set and IS. Finding the two objects $r.delete$ and $r.candidate$ requires just a simple scan on O_{knn} and the IS.

In a metric space where the distance computation is simple, e.g., Euclidean space and weighted Euclidean space, we can just compare the distance from the query object to $r.candidate$ and $r.delete$ as discussed above. In spaces where the distance computation is non-trivial, e.g., spatial networks and obstructed spaces, we need more efficient algorithms for the validation.

In a spatial network or an obstructed space, the method described above still gives correct answers. However, a simple scan on the two sets to compute the distances of the objects to q may be expensive because it requires graph shortest path computation. We propose an algorithm to reduce the validation cost as summarized in Algorithm 3. We run a procedure similar to the VN^3 algorithm [13], but on a subset of the network (note the algorithm input). The algorithm keeps exploring the network to find the new kNN. Once any NN found is not in the current kNN set O_{knn}, the algorithm can terminate early (Lines 1 and 8). Essentially, this is to recompute the kNN set, but on a much smaller network consisting of only edges from Voronoi cells of objects in O_{knn} and $I(O_{knn})$. An important advantage is that this smaller network can be sent to the query user together with the kNN set, which allows the query user to validate the kNN set locally rather than requiring the query processor to validate whenever the query object changes its location.

The following theorem guarantees that the shortest distances computed on this partial network are identical to those computed on the full network.

Theorem 3. *Given a spatial network Voronoi diagram D_O on a set of data objects O, a* kNN *set $O_{knn} \subset O$ and its INS $I(O_{knn})$, and a Voronoi diagram $D_{O_{knn} \cup I(O_{knn})}$ formed by the edges and vertices from the Voronoi cells of the objects in O_{knn} and $I(O_{knn})$, if the* kNN *set of a query object q is O_{knn} on $D_{O_{knn} \cup I(O_{knn})}$, then the* kNN *set of q on D_O is also O_{knn}.*

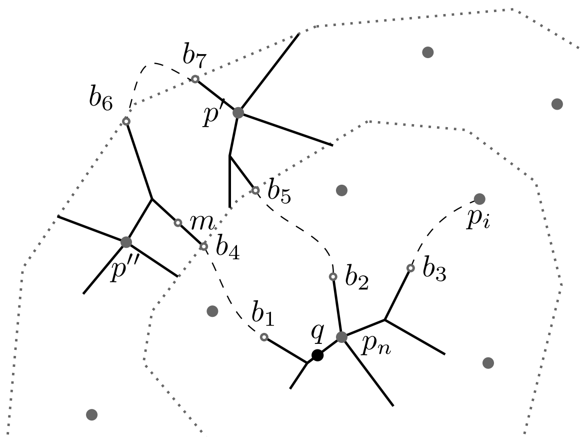


Fig. 7. Theorem 3.

Proof. We use Fig. 7 to illustrate the proof. In this figure, the gray dots denote the data objects and the solid lines connected to an object constitute the network Voronoi cell of the object. For simplicity we only show the network Voronoi cells of three objects p_n, p', and p''. The paths among them that are within the non-depicted cells are denoted by the dashed lines. The area between the inner dotted circle and outer dotted circle contains the Voronoi cells of $I(O_{knn})$. Note that the dotted circles are not part of the network. They are only used to separate the Voronoi cells conceptually.

We use $P_O(p_i, p_j)$ and $P_{O_{knn} \cup I(O_{knn})}(p_i, p_j)$ to denote the shortest path from p_i to p_j on D_O and $D_{O_{knn} \cup I(O_{knn})}$, respectively. We use $d_O(p_i, p_j)$ and $d_{O_{knn} \cup I(O_{knn})}(p_i, p_j)$ to denote the respective shortest path distances, and $V(p_i)$ to denote the network Voronoi cell of p_i.

We prove by contradiction. Suppose that the kNN set of q on D_O is not O_{knn}. By the definition of influential set (Definition 3) and Theorem 2, a data object $p' \in I(O_{knn})$ must be nearer to q than an object $p_i \in O_{knn}$, i.e., $d_O(q, p') < d_O(q, p_i)$. The path $P_O(q, p')$ must contain some shortcut that does not belong to $D_{O_{knn} \cup I(O_{knn})}$, otherwise $d_O(q, p')$ should be equal to $d_{O_{knn} \cup I(O_{knn})}(q, p')$, and not smaller than $d_{O_{knn} \cup I(O_{knn})}(q, p_i)$. In Fig. 7, let $P_{O_{knn} \cup I(O_{knn})}(q, p')$ be the path from q to p' through b_2 and b_5, and $P_O(q, p')$ be the path from q to p' through b_1, b_4, b_6, and b_7. Assume that the latter path is shorter and we derive a contradiction.

Let p_n be the nearest neighbor of q. Then $p_n \in O_{knn}$, and q is in $V(p_n)$. The shortest path from q to $p' \in I(O_{knn})$ must pass through the Voronoi cell of some object $p'' \in I(O_{knn})$ since the Voronoi cells of O_{knn} is surrounded by the Voronoi cells of the objects in $I(O_{knn})$. There are two cases of p'':

(i) $P_O(q, p'') = P_{O_{knn} \cup I(O_{knn})}(q, p'')$. Let m be a point on $P_O(q, p')$ that is in $V(p'')$. By the definition of Voronoi cell, $d(m, p'') < d(m, p')$, and hence $d_O(q, p'') \le d_O(q, m) + d_O(m, p'') < d_O(q, m) + d_O(m, p') = d_O(q, p')$. Meanwhile, the kNN set of q on $D_{O_{knn} \cup I(O_{knn})}$ is O_{knn}. Thus, $d_O(q, p'') = d_{O_{knn} \cup I(O_{knn})}(q, p'') > d_{O_{knn} \cup I(O_{knn})}(q, p_i) \ge d_O(q, p_i) > d_O(q, p')$. We hence obtain a contradiction.

(ii) $P_O(q, p'') \ne P_{O_{knn} \cup I(O_{knn})}(q, p'')$. In this case, $P_O(q, p'')$ also contains edges outside $D_{O_{knn} \cup I(O_{knn})}$. We treat p'' as p' and find another object p''' in $I(O_{knn})$ as the new p'' to satisfy Case (i). Since $P_O(q, p')$ has a finite number of objects, by repeating this process we can eventually find an object satisfying Case (i) and obtain a contradiction.

5.2. kNN set update

When the kNN set becomes invalid, we need to update it and the corresponding IS. If the kNN set is not fully contained by R, we need to update R and its INS, $I(R)$.

Algorithm 4 Updating

```
Input: query object $q$, prefetched set $R$, current $k$NN set $O_{knn}$ and
    its influential set $I$, $r.candidate$, $r.delete$
Output: new prefetched set $R$, $k$NN set $O'_{knn}$ and its influential set
    $I'$
1:  $O'_{knn} \leftarrow O_{knn} \backslash \{r.delete\} \cup \{r.candidate\}$
2:  $I' \leftarrow (I \cup \{r.delete\} \cup N_O(r.candidate)) \backslash O'_{knn}$
3:  if $O'_{knn} \prec_q I'$ then
4:      return $(R,\ O'_{knn},\ I')$
5:  else
6:      $O'_{knn} \leftarrow kNN$ in $R$, $I' \leftarrow I \cup O_{knn} \backslash O'_{knn}$
7:      if $O'_{knn} \prec_q I'$ then
8:          return $(R,\ O'_{knn},\ I')$
9:      else
10:         Recompute $R$, $I(R)$
11:         $O'_{knn} \leftarrow$ top $k$ in $R$, $I' \leftarrow R \cup I(R) \backslash O'_{knn}$
12:         return $(R,\ O'_{knn},\ I')$
```

In general, a simple scan on the data set with the support of a size-$\lfloor \rho k \rfloor$ queue of the data objects prioritized by their distances to the query object is sufficient to identify the new $\lfloor \rho k \rfloor$ NN set. A metric space index such as the M-tree [17] may help further reduce the $\lfloor \rho k \rfloor$ NN set computation complexity. Then the corresponding INS can be retrieved from the precomputed order-1 Voronoi diagram.

In a spatial network or in an obstructed space, we simply run the VN^3 algorithm to compute the new $\lfloor \rho k \rfloor$ NN set. The INS is then retrieved from the precomputed order-1 Voronoi diagram. We omit the pseudo code as it is straightforward.

In a space where the distance computation is simple, e.g., Euclidean space and weighted space, the method described above still applies. However, we can obtain R and $I(R)$ more efficiently. We consider two cases: (i) If q has entered a neighboring order-k Voronoi cell, the new kNN set will differ from the current one by only one data object, since the boundary just crossed must be shared by two objects, one in O_{knn} and the other in the IS. In this case, we do not need to recompute the entire kNN set but can use the existing kNN set to compose the new kNN set. (ii) If q is not in a neighboring order-k Voronoi cell, we recompute the kNN set and check whether it is still in R. If yes, we just need to return this set. If not, we compute and return the new sets of R and $I(R)$.

Since we do not compute order-k Voronoi cells, we cannot determine whether q has entered a neighboring order-k Voronoi cell. However, we observe that, when the kNN set becomes invalid, there is an object $r.delete$ to be removed from the current kNN set and an object $r.candidate$ to be added to the new kNN set. This gives us a candidate new kNN set $O'_{knn} = O_{knn} \cup \{r.candidate\} \backslash \{r.delete\}$. We test whether it is indeed the new kNN set by testing whether it is closer to q than its IS.

According to Definition 5, the INS of O'_{knn} consists of the Voronoi neighbors of all objects in O'_{knn}. We know that O_{knn} only differs from O'_{knn} by $\{r.candidate, r.delete\}$. We can derive an IS of O'_{knn} from the IS of O_{knn} as $I(O'_{knn}) = I(O_{knn}) \cup \{r.delete\} \cup N_O(r.candidate) \backslash O'_{knn}$. Then we test whether $I(O'_{knn}) \circlearrowleft O'_{knn}$ holds. (i) If yes, O'_{knn} is the new kNN set. We return it and its IS. (ii) If not, we recompute the kNN set. Algorithm 4 summarizes the procedure.

Note that we have precomputed the Voronoi diagram of O and stored every object with its Voronoi neighbors. When sending the set R to the query user, we also send the corresponding Voronoi neighbors. Thus, no communication cost will incur unless the Voronoi neighbors of $r.candidate$ need to be added to IS (Line 2), or the sets R and $I(R)$ need to be recomputed (Lines 8 and 9).

Algorithm 5 Insertion

```
Input: new data object $p$, prefetched set $R$, query object $q$, $k$NN
    set $O_{knn}$ and its influential set $IS$
Output: updated prefetched set $R$, $k$NN set $O_{knn}$ and influential
    set $I$
1:  if not $R \prec_q \{p\}$ then
2:      $R \leftarrow R \cup \{p\}$
3:      $p^k \leftarrow$ the $k^{th}$ nearest neighbor of $q$ in $O_{knn}$
4:      if $d(p, q) < d(p^k, q)$ then
5:          return $(R,\ O_{knn} \cup \{p\} \backslash \{p^k\},\ I \cup \{p^k\})$
    return $(R,\ O_{knn},\ I \cup \{p\})$
```

5.3. Data object update

When there are data object updates, we first update the Voronoi diagram and the index structures, and then update the kNN set and the IS. For Voronoi diagram and index update, space specific algorithms are needed. We use the R*-tree update algorithm [14] and the VoR-tree update algorithm [16] in the Euclidean space, and run the VN^3 algorithm [13] locally to update the Voronoi cells and the related data structure in spatial networks.

Next, we detail the kNN set and IS update procedure. We focus on insertion and deletion, as a location update can be done by deletion and re-insertion.

5.3.1. Insertion

When a new data object p is added, there are two cases: (i) if it is nearer to the query object q than a current NN, we add it to the kNN set and update IS using the current kNN set, the current IS and p; (ii) otherwise, we simply add p to the IS (Theorem 1 guarantees the correctness).

Note that in the above insertion procedure, we do not add the Voronoi neighbors of p to IS when p is added to O_{knn}. Instead, we add p^k, which is the kth NN of q in O_{knn}, to IS. This is supported by the following theorem.

Theorem 4. *Given a query object q, its current kNN set O_{knn} where p^k is the kth nearest neighbor, and a new data object p satisfying $d(p, q) < d(p^k, q)$, for the new kNN set $O'_{knn} = O_{knn} \cup \{p\} \backslash \{p^k\}$, we have:*

$$I(O_{knn}) \cup \{p^k\} \circlearrowleft O'_{knn}. \tag{15}$$

Proof. The proof in the conference paper [10] still applies in metric space and is omitted here.

Algorithm 5 summarizes the process. The algorithm takes a sequential scan on R with $\mathcal{O}(|R|)$ comparisons to determine whether the condition in Line 1 holds. The rest of the algorithm takes a constant time. Therefore, the complexity of the algorithm is linear to the cardinality of R.

5.3.2. Deletion

When a data object p_i is to be deleted, we first check if it belongs to R or $I(R)$. If so, we delete it from R or $I(R)$ (Lines 1 and 2) and consider the following three cases: (i) p_i is in the kNN set; (ii) p_i is in the IS; (iii) p_i is not in either the kNN set or the IS. Case (iii) does not involve kNN set or IS update and hence is not discussed further.

(i) Deletion from the kNN set: If $p_i \in O_{knn}$, both O_{knn} and IS need to be updated (Lines 3 to 6). We find the nearest data object in the IS to replace p_i in O_{knn} (Line 6), the correctness of which is guaranteed by the following theorem.

Algorithm 6 Deletion

```
Input: data object to be deleted $p_i$, prefetched set $R$, query object $q$, current $k$NN set $O_{knn}$ and its influential set $I$
Output: updated prefetched set $R$, $k$NN set $O_{knn}$ and influential set $I$
1:  if $p_i \in R$ or $p_i \in I$ then
2:      $R \leftarrow R \backslash \{p_i\}$, $I \leftarrow I \backslash \{p_i\}$
3:      if $p_i \in O_{knn}$ then
4:          $p_j \leftarrow$ nearest neighbor to $q$ in $I$
5:          $I' \leftarrow I \cup N_O(p_j) \backslash O_{knn} \backslash \{p_j\}$
6:          return $(R, O_{knn} \backslash \{p_i\} \cup \{p_j\}, I')$
7:      else if $p_i \in I$ then
8:          return $(R,\ O_{knn},\ I \cup N_O(p_i) \backslash O_{knn} \backslash \{p_i\})$
9:  else
10:     return $(R,\ O_{knn},\ I)$;
```

Theorem 5. *Given a* k*NN set* O_{knn} *and a set* $IS \supseteq I(O_{knn})$*, if a data object* p *in* O_{knn} *is deleted, then the new* k*NN set* O'_{knn} *should be*

$$O'_{knn} = O_{knn} \backslash \{p_i\} \cup \{p_j\},\ p_j \in IS. \tag{16}$$

Proof. See the conference paper [10]. We omit the proof here due to space limit.

After p_j is added to O'_{knn}, according to Definition 5, its Voronoi neighbors are added to the IS (Line 5).

(ii) Deletion from the IS: If p_i belongs to the IS, we update the IS by first removing p_i and then adding all Voronoi neighbors of p_i which are not currently in O_{knn} or the IS to the IS (Line 8), as supported by the following theorem.

Theorem 6. *Given a* k*NN set* O_{knn} *and its IS, if a data object* $p_i \in IS$ *is deleted, we have*

$$IS \cup N_O(p_i) \backslash O_{knn} \backslash \{p_i\} \circlearrowleft O_{knn}. \tag{17}$$

Proof. Intuitively, when $p_i \in IS$ is to be removed, we need its Voronoi neighbors to replace p_i to safe guard the kNN set. We omit the full proof due to space limit.

Processing of Cases (i) and (ii) is summarized in Algorithm 6. The algorithm takes a sequential scan on R and I to determine whether the conditions in Lines 1, 3, and 7 hold, which takes $\mathcal{O}(|R| + |I|)$ time. The rest of the algorithm takes a constant time. Therefore, the complexity of the algorithm is linear to the cardinality of R and I.

5.4. Keyword query processing

Our method can be extended to process spatial keyword queries using weighted Euclidean distance. Given a set W of keywords, each object $o \in O$ is assigned a set of keywords $o.kw$, where $o.kw \subseteq W$. Define a similarity function $\zeta(o, q)$, let $q.kw$ be a set of keywords. The similarity function can be defined based on the keyword similarity between o and q, e.g., if $o.kw = \{spaghetti, macaroni\}$ and $q.kw = \{macaroni, sushi\}$, the similarity between o and q can be defined as 0.5, i.e., 50% of the keywords in o.kw are in q.kw. Then, the weighted distance between the query q and object o is defined as $d(o, q)/\zeta(o, q)$, where $d(o, q)$ is the Euclidean distance between o and q. Here, if $\zeta(o, q) = 0$, which means there is no overlap between the query and the object, we define the weighted distance as ∞. Since weighted Euclidean space is a generalized metric space, our method applies straightforwardly.

As a special case, if $\zeta(o, q)$ returns 1 when $o.kw \supset q.kw$ and 0 if $o.kw \not\supset q.kw$, we can use our solution for the Euclidean space to save computation costs. We pre-compute $|W|$ order-1 Voronoi diagrams, one for each keyword in W. For a diagram with regard to keyword $w \in W$, denoted by D_w, it only contains objects whose keyword sets contain w, i.e., for any object o' in D_w, $w \in o'.kw$. As a result, the $|W|$ diagrams pre-computed require a total of $\mathcal{O}(\sum_{o \in O} |o.kw|)$ space.

To process a continuous keyword query q that contains a keyword set $q.kw$, we keep track of the kNN result for each keyword w in $q.kw$ individually and merge the $|q.kw|$ kNN result sets together to produce the final kNN results. The overall computation time is $|q.kw|$ times the time of a single continuous kNN query, plus $\mathcal{O}(|q.kw|k \log |q.kw|k)$ which is for merging the $|q.kw|$ kNN result sets.

6. Cost analysis

There is no universal cost analysis that applies across all metric spaces since different kNN algorithms may be used. We have analyzed the costs of the proposed Euclidean space MkNN algorithm in the previous conference paper [10]. Here, we present a comparative cost analysis on the spatial network MkNN algorithm (denoted by INS-kNN) and the V*-Diagram algorithm [12]. Following V*-Diagram [12], we assume uniform data distribution, and that a query processor holds the set of data objects and reports kNN sets to a query user when required.

***k*NN set recomputation frequency:** Let the kNN set recomputation frequency of INS-kNN and V*-Diagram be f_{INS} and f_{VD}, respectively. The recomputation frequency is mainly affected by the query object speed and the size of the safe regions (which is determined by the density of data objects). Following V*-Diagram [12], we assume a constant query object speed and analyze the impact of the size of the safe regions. This analysis is very similar to that in the Euclidean space [10]. The only difference is that now the recomputation frequency is inversely proportional to the total length of the network edges in the Voronoi cells of R (while in the Euclidean space it is inversely proportional to the square root of the area of the Voronoi cells).

Let the total length of all the edges in a spatial network be 1. According to Okabe et al. [7], the total length of the edges in the Voronoi cells of R is $\mathcal{O}(k(\lfloor \rho k \rfloor - k)/(k(n-k))) = \mathcal{O}((\lfloor \rho k \rfloor - k)/(n-k))$, where n denotes the number of data objects. Thus, $f_{\text{INS}} = \mathcal{O}((n-k)/(\lfloor \rho k \rfloor - k))$. On the other hand, according to [12], $f_{\text{VD}} = \mathcal{O}(nk/x)$. Therefore, $f_{\text{INS}}/f_{\text{VD}} = \mathcal{O}((n-k)/nk) < \mathcal{O}(1/k)$.

Communication cost: When the kNN set is recomputed, the new query result and safe region need to sent to the query user. INS-kNN sends $\mathcal{O}(\lfloor \rho k \rfloor)$ objects while V*-Diagram sends $\mathcal{O}(k + x)$ objects each time. Since $\lfloor \rho k \rfloor$ in INS-kNN essentially plays the same role as that of $k + x$ in V*-Diagram, the communication cost of the two methods is determined by their kNN set recomputation cost.

***k*NN set recomputation cost:** In Algorithm 1 for kNN computation, INS-kNN calls function `Contain()` for one time and function `ShortestDistance()` for $\lfloor \rho k \rfloor$ times. Function `Contain()` takes $\mathcal{O}(\log n + \lfloor \rho k \rfloor)$ time to get the (MBR of the) Voronoi cell containing the query object from an R*-tree [14]. Function `ShortestDistance()` computes the network shortest path distance. It can be implemented in an incremental manner, i.e., the intermediate result obtained from one function call is saved and used for the following function calls. Then essentially, the $\lfloor \rho k \rfloor$ times of the function calls together can be seen as one call of the Dijkstra's algorithm [18] to find the $\lfloor \rho k \rfloor$ nearest neighbors.

It is shown [7] that the average number of Voronoi neighbors per object (and hence average number of edges per Voronoi cell) is a small constant less than 6. Thus, Algorithm 1 only involves $\mathcal{O}(\lfloor \rho k \rfloor)$ edges. Running Dijkstra's algorithm to find the $\lfloor \rho k \rfloor$

nearest neighbors from $\mathcal{O}(\lfloor\rho k\rfloor)$ edges takes $\mathcal{O}(\lfloor\rho k\rfloor\log(\lfloor\rho k\rfloor))$ time [18]. Therefore, the *k*NN set recomputation of INS-*k*NN takes $\mathcal{O}(\log n+\lfloor\rho k\rfloor\log(\lfloor\rho k\rfloor))$ time.

On the other hand, the recomputation cost of V*-Diagram consists of three steps: (i) finding the first $k+x$ nearest neighbors, (ii) computing the safe region w.r.t. the *k*th nearest neighbor, and (iii) computing the FRR. These three steps need $\mathcal{O}(\log n+k+x)$, $\mathcal{O}((k+x)\log((k+x)))$, and $\mathcal{O}((k+x)^2)$ time, respectively [12]. The overall recomputation cost of V*-Diagram is $\mathcal{O}(\log n+(k+x)^2)$.

***k*NN set validation cost:** The algorithm for *k*NN validation, Algorithm 3, has a procedure similar to that of Algorithm 1, but runs on a smaller network of $\mathcal{O}(\lfloor\rho k\rfloor)$ edges. Thus, INS-*k*NN needs $\mathcal{O}(\lfloor\rho k\rfloor\log(\lfloor\rho k\rfloor))$ time for the validation. Meanwhile, V*-Diagram needs to check whether the query object has left the current safe region by checking whether it has passed through any of the safe region boundary points, which takes $\mathcal{O}\left((k+x)^2\right)$ time [12].

In summary, INS-*k*NN has smaller costs comparing with V*-Diagram in the above analysis.

Discussion The above analysis shows that the recomputation and validation costs of our method in spatial networks are $\mathcal{O}(\log n+\lfloor\rho k\rfloor\log(\lfloor\rho k\rfloor))$ and $\mathcal{O}(\lfloor\rho k\rfloor\log(\lfloor\rho k\rfloor))$, respectively. In our previous conference paper [10], we show that the corresponding costs are $\mathcal{O}(\log n+\lfloor\rho k\rfloor)$ and $\mathcal{O}(\lfloor\rho k\rfloor)$ in Euclidean space. The differences in the costs come from the cost differences in nearest neighbor distance computation in the two types of spaces. In spatial networks, we run Dijkstra's algorithm to find the shortest path and obtain the shortest distance. This procedure requires more time than computing the Euclidean distance in Euclidean space. Therefore, if we query the two datasets of the same size but in these two types of spaces, the query cost in spatial networks is most likely to be higher.

In other metric spaces, the costs are different due to the different costs of distance computation between objects. In an obstructed space, for example, we run Dijkstra's algorithm on the visibility network to find nearest neighbors of a query. The cost of our method in an obstructed space is the same as that in the spatial network. On the other hand, in a weighted space, since the distances between objects and the query can be computed directly from points and weights of objects, the cost of our method in weighted space is the same as that in the Euclidean space.

7. Demonstration

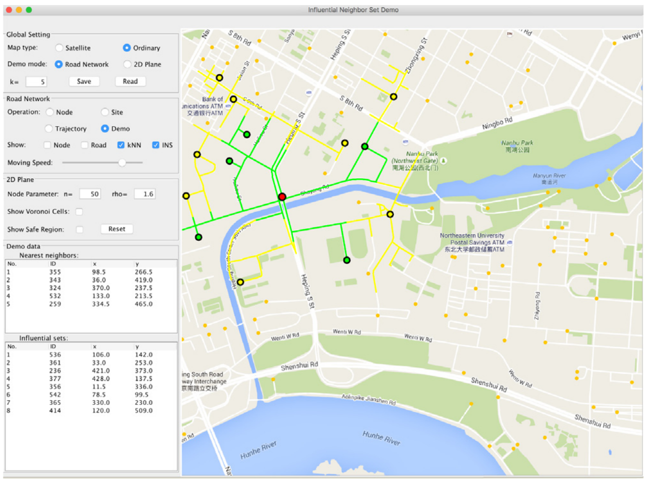
We build an interactive demonstration system that shows the internal mechanism of the INS algorithm. It simulates a moving query object and displays the changing *k*NNs as well as the safe guarding objects continuously. The users are given options to customize several parameters for information and aesthetic purposes such as the number of nearest neighbors displayed.

The system is implemented as a Scala Swing application. It runs in two modes, *Spatial Network* mode and *2D Plane* (Euclidean space) mode. Its user interface consists of two panels (cf. Fig. 8):

(i) *Control panel.* This is the left panel of the user interface. It is used for setting the demonstration parameters, which include the *Global Setting*, the *Spatial Network* setting and the *2D Plane* setting.

The global setting includes the underlying map to be used, the data space to be used, and the value of the query parameter *k*. There are also two buttons "Save" and "Read" which are for recording and loading the demonstration settings.

The spatial network setting includes options to control the operations being performed: "*Node*", "*Site*", "*Trajectory*", and "*Demo*". When one of the first three options is chosen, it allows users to add/move/delete the network nodes/data objects/query trajectory. When "Demo" is chosen, a M*k*NN query will be simulated. The setting also includes options to control which objects



Global Setting
Map type: Satellite [✓ Ordinary]
Demo mode: [✓ Road Network] 2D Plane
k= 5

Road Network
Operation: Node Site Trajectory [✓ Demo]
Show: Node Road [✓ kNN] [✓ INS]
Moving Speed:

2D Plane
Node Parameter: n= 50 rho= 1.6
Show Voronoi Cells:
Show Safe Region:

Demo data
Nearest neighbors:

No.	ID	x	y
1	355	98.5	266.5
2	343	36.0	419.0
3	324	370.0	237.5
4	532	133.0	213.5
5	259	334.5	465.0

Influential sets:

No.	ID	x	y
1	536	106.0	142.0
2	361	33.0	253.0
3	236	421.0	373.0
4	377	428.0	137.5
5	356	11.5	336.0
6	542	78.5	99.5
7	365	330.0	230.0
8	414	120.0	509.0

Fig. 8. Interface of the demo system . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

to be shown as well as the query object's moving speed in the simulation.

The 2D plane setting includes the number of data objects to generate (n), the value of the prefetch ratio (ρ), and whether to show the Voronoi cells and the corresponding safe regions.

Under these settings, the current *k*NN set and the INS are displayed.

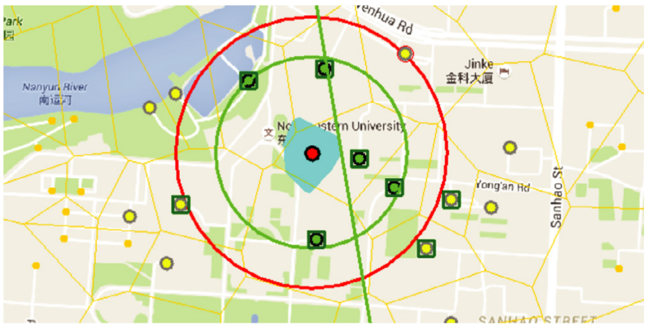
(ii) *Main panel.* This is the right panel of the user interface. It displays a map of a spatial network where data objects (orange dots) and the query object (red dot) can be placed onto. Users can specify a trajectory for the query object. In the Spatial Network mode, this trajectory must confine the underlying spatial network; in the 2D Plane mode, the trajectory can have any shape. When query simulation starts, the query object will move along the specified trajectory. The INS algorithm will run to compute the *k*NN set and the INS set, which are shown as the green dots and the yellow dots, respectively.

In the Spatial Network mode, the Voronoi cells of the objects in the *k*NN set and the INS are represented by the sets of green and yellow network edges, respectively.

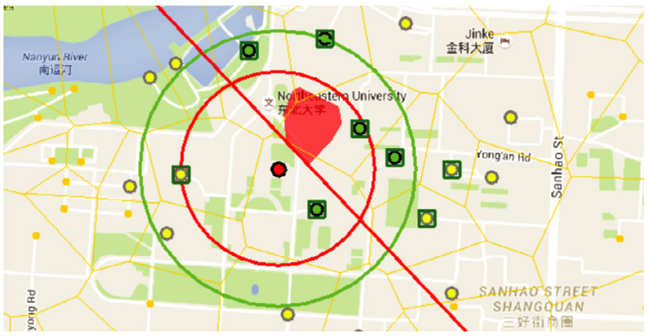
In the 2D Plane mode (Fig. 9), the data objects are surrounded by their respective order-1 Voronoi cells. The ones enclosed by green squares represent the objects in *R*. We use the cyan polygon to represent the current order-*k* Voronoi cell, which will turn red when it becomes invalid. We circle two special objects, the farthest object to *q* in the *k*NN set and the nearest object to *q* in the INS, with a green circle and a red circle passing through them, respectively, where the center of both circles are at *q*. These two objects are special in the sense that the green circle should always enclose all the objects in the *k*NN set, while the red circle should never enclose any object in the INS, as long as the current *k*NN set is still valid.

Demonstrated Scenarios: We take the 2D Plane mode as an example. Fig. 9 displays two screenshots of the demonstration program, where each shows: (i) the query object stays in the order-*k* Voronoi cell of the current *k*NN set (i.e., the *k*NN set is valid), and (ii) the query object has moved out of the order-*k* Voronoi cell (i.e., the *k*NN set is invalid).

When the query object moves out of the order-*k* Voronoi cell shown in Fig. 9(a) and moves into the position shown in Fig. 9(b), the current *k*NN set becomes invalid since the green circle is now enclosed by the red circle (i.e., the farthest object to *q* in the *k*NN set is farther to *q* than the nearest object to *q* in the INS).



(a) The kNN set is valid



(b) The kNN set is invalid

Fig. 9. kNN and safe guarding objects . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Euclidean space experimental settings.

Parameter	Default	Values
x, number of auxiliary objects	6	$1, 2, \ldots, 19$
k, number of queried objects	10	$1, 2, \ldots, 19$
ρ, prefetch ratio	1.7	$1.1, 1.2, \ldots, 2.5$
Speed of the query objects (m/timestamp)	20	$2, 4, \ldots, 36$
Trajectory length (km)	10	$1, 2, \ldots, 16$
Object-vertex ratio (%)	10	$5, 5.5, \ldots, 10.5$

8. Experiments

We compare empirically our algorithm with the state-of-the-art algorithm V*-Diagram [2] as described in Section 2.4. V*-Diagram works in both Euclidean space and spatial networks. We have previously reported the experimental results in Euclidean space [10]. Here, we present the results in spatial networks. We additionally compare with the *Local Network Voronoi Diagram* (LNVD) algorithm [19] which computes a local network Voronoi diagram within a small range around the query object as the safe regions. Following the original proposal [19], we set the search range of LNVD at $3k/r$, where r is the object-vertex ratio, i.e., the number of data objects divided by the number of network vertices.

In the result figures we use **V***, **LN**, and **INS** to denote V*-Diagram, LNVD, and the proposed algorithm, respectively.

The algorithms are implemented using C++ (gcc 5.2 -O3) on a computer running CentOS 7 with two 2.5 GHz Intel Xeon E5 CPU and 16 GB memory.

8.1. Settings

We use a real spatial network, the full USA road network,[2] in the experiments. It contains 23,947,347 vertices and 58,333,344 edges. The vertices are randomly chosen to form data object sets of different sizes. We generate two types of query objects: *random (denoted by "**R**")* and *directional (denoted by "**D**")*. A random query object starts at a randomly chosen vertex and moves to a randomly chosen edge every time it reaches a vertex; a directional query object starts at a randomly chosen vertex and moves to a randomly chosen destination vertex along the shortest path. We vary the values of the parameters as summarized in Table 2.

We assume a client–server model, where the server holds the data set in memory and computes the kNN sets, while the clients are the query users and they keep the current kNN sets and IS. At every timestamp, the kNN set is first validated on the client side. A result update request is sent to the server if the kNN set has become invalid.

We record the response time (on both server side and client side) and the communication cost between the query user and the server. Here, the communication cost is counted as the number of objects transferred from the server to the query user. We run each experiment 20 times and report the average total cost of processing a query.

8.2. Results

We show the server response time, client response time, and communication costs in Figs. 10, 11, and 12, respectively.

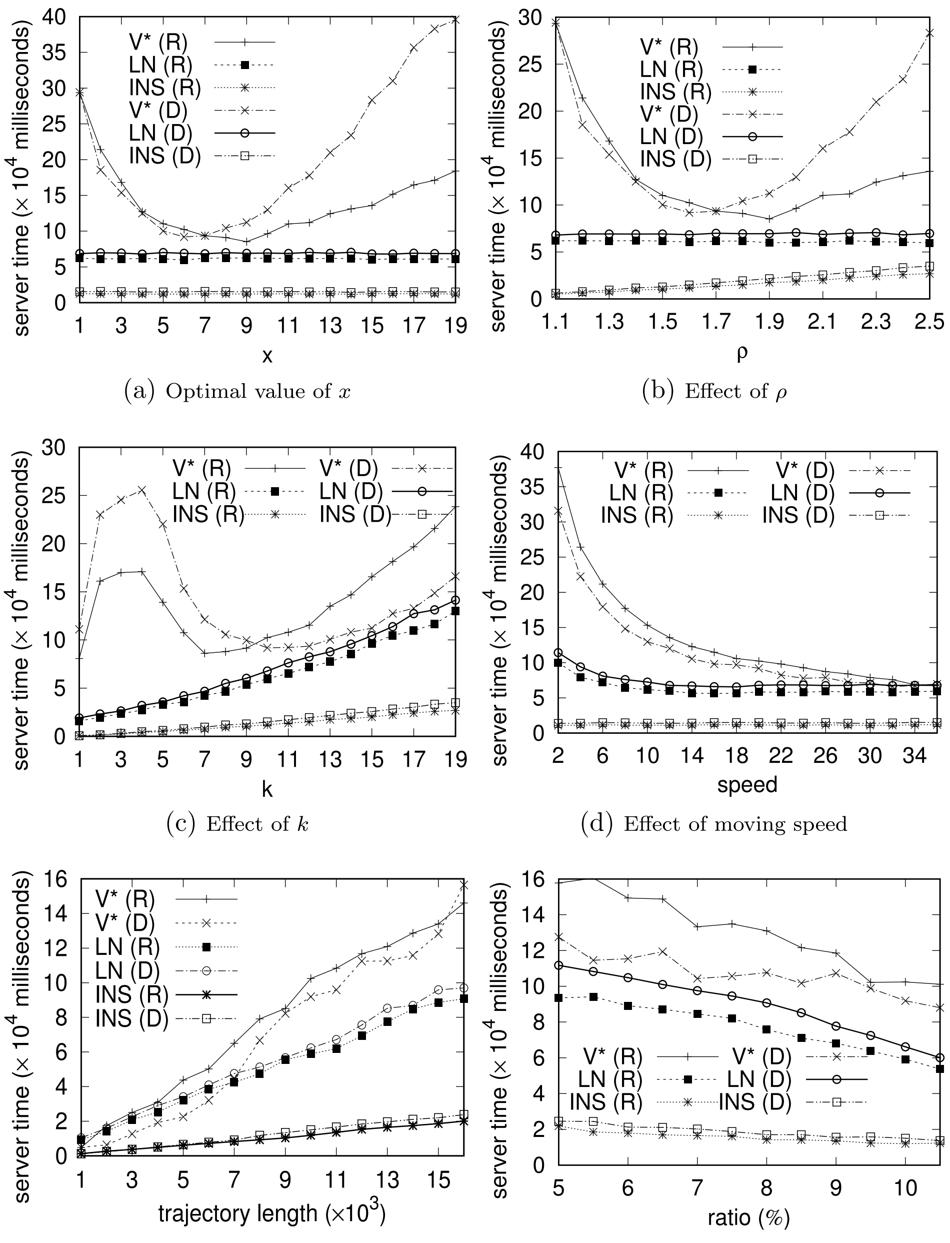
Parameter optimization for V*-Diagram. V*-Diagram has a parameter x, the number of auxiliary data objects used in safe region computation. We first find the optimal value of x empirically. Figs. 10∼12(a) show the algorithm performance for x ranging from 1 to 19, where other parameters are set to the default values. We observe that the costs of V*-Diagram first drop when x increases, and then increase again as x becomes larger. When $x = 6$, V*-Diagram shows the best overall performance. This has been observed by Nutanong et al. [2] and can be explained by that (i) a small number of auxiliary data objects can help reduce the recomputation frequency, but (ii) a larger number of auxiliary data objects incur too much extra costs which outweighs the benefits. Experiments on other settings show a similar pattern and hence in what follows we use $x = 6$ as the default x value.

Neither LNVD nor INS uses the parameter x, and hence their performance is not affected. We see that even when $x = 6$ where V*-Diagram shows the best performance, INS still outperforms V*-Diagram. This is because the conceptual safe region used in INS is guaranteed to be the largest possible while the safe region used in V*-Diagram is not.

Effect of ρ. INS uses a parameter ρ ($\rho \geq 1$) that plays a similar role to that of x in V*-Diagram, i.e, to balance the computation and communication costs. We compute $\lfloor \rho k \rfloor$ NNs and use the extra ($\lfloor \rho k \rfloor - k$) objects as a kNN "cache". We evaluate the effect of ρ by varying its value from 1.1 to 2.5. We set the value of x in V*-Diagram to be $\lfloor \rho k \rfloor - k$ so that both algorithms have the same size of kNN "cache".

Figs. 10∼12(b) show the result. INS outperforms V*-Diagram and LNVD in both server response time and communication cost under the various values of ρ tested, while LNVD shows a lower client response time. As expected, when ρ increases, the communication cost of INS decreases because now it requires less kNN set transmission from the server to the client. Meanwhile, the other costs of INS increase because both the server side kNN recomputation and the client side kNN validation now have a

2 http://www.dis.uniroma1.it/challenge9/download.shtml.



(a) Optimal value of x (b) Effect of ρ

(c) Effect of k (d) Effect of moving speed

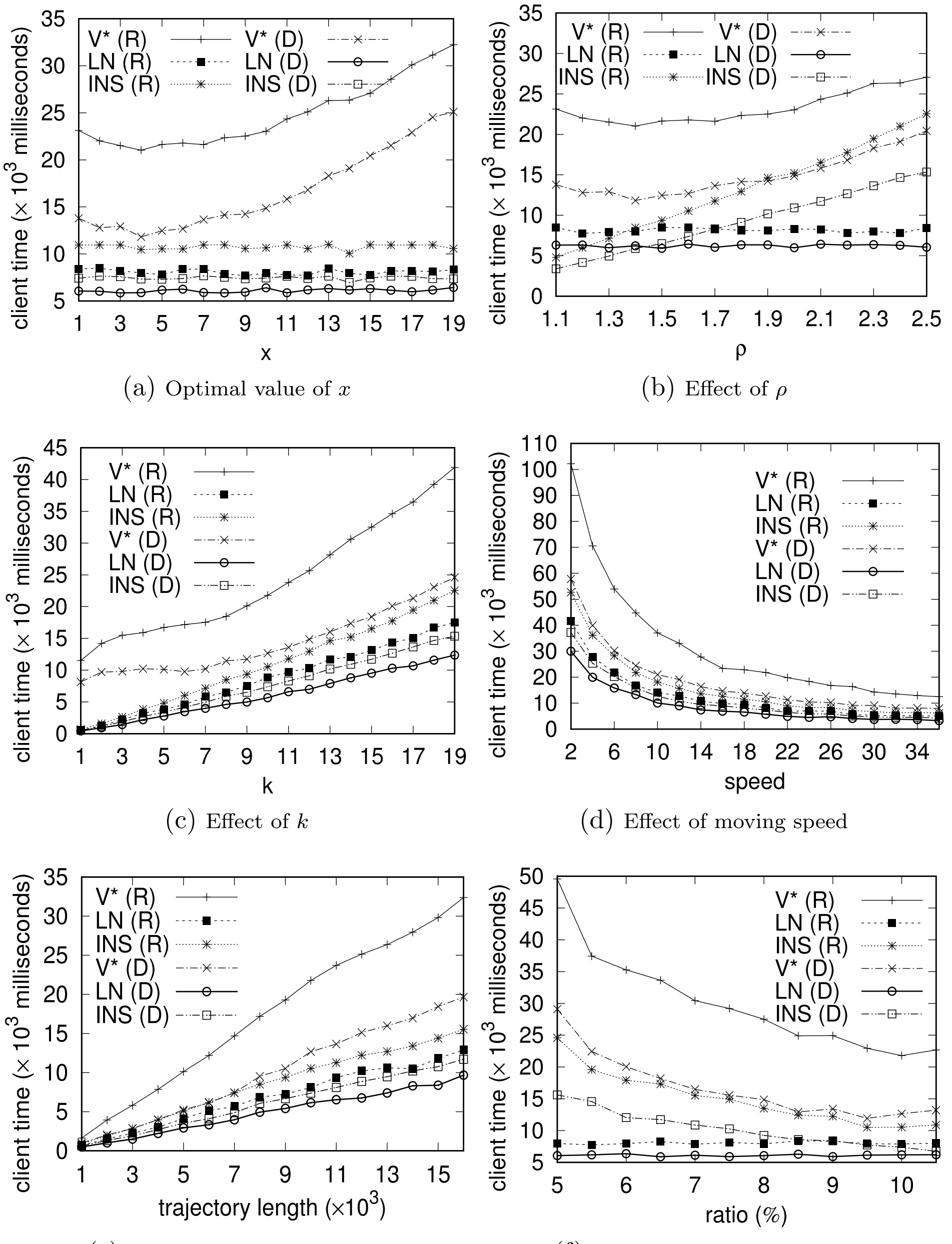
(e) Effect of trajectory length (f) Effect of data object density

Fig. 10. Server response time.

larger $\lfloor \rho k \rfloor$ NN set to process. The client response time has a larger increase in particular. This is because now a larger sub-network needs to be traversed when validating the kNN sets. However, we argue that this is not a significant drawback because the total client response time is still quite small (less than 2.5 s for all the values tested). To obtain an overall better performance, we use $\rho = 1.7$ as the default value in the following experiments.

Varying *k*. Next we vary the value of k from 1 to 19. Figs. 10∼12(c) show the results. We see that while the costs of V*-Diagram and LNVD increase rapidly with the value of k, those of INS increase much slower. This is because INS computes the largest safe region possible and utilizes the current kNN set to derive the new kNN set. It keeps the number of kNN computation to the minimum. Note that V*-Diagram has a higher cost when $k < 7$. This is because by default 6 auxiliary objects are computed for V*-Diagram. The costs of computing these extra auxiliary objects outweigh the costs necessary for computing the kNN sets. INS avoids this by computing $\lfloor \rho k \rfloor$ NNs. When k is small, the number of extra NNs computed is small as well.

Varying query object speed. Figs. 10∼12(d) show the algorithm performance when the query object speed increases from 2 to 36 m per timestamp. As the speed increases, the server and client response time of all three algorithms decrease. The reason is as follows. The client side validates the kNN set at every timestamp, and the server side recomputes the kNN set when necessary. As the speed increases, the total number of timestamps required to complete a query trajectory decreases which means that both the recomputation and validation costs will decrease. Meanwhile, the communication costs of the algorithms increase. This is because we have used speed values and query trajectories that guarantee that the query object does not cross more than one order-k Voronoi cell within one timestamp, i.e., all the kNN set changes are captured in the query processing procedure. With a high speed the query object moves outside the safe region more frequently, and updated query result needs to be sent to the query



(a) Optimal value of x (b) Effect of ρ

(c) Effect of k (d) Effect of moving speed

(e) Effect of trajectory length (f) Effect of data object density

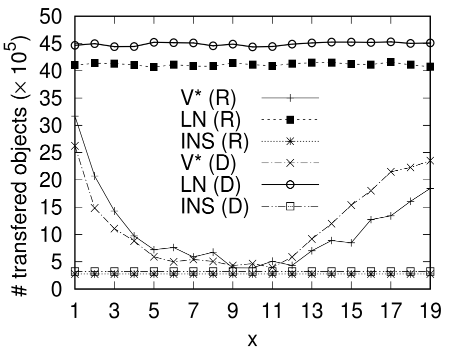
Fig. 11. Client response time.

object more frequently. INS has the smallest communication cost due to its larger safe regions.

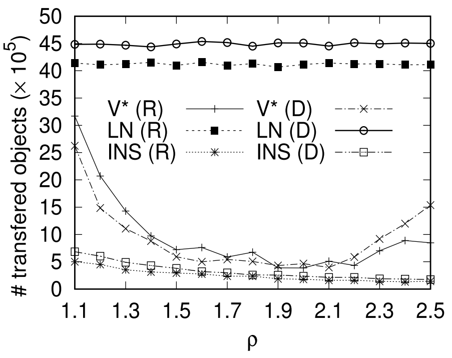
Varying query trajectory length. When the query trajectory length increases, the overall query processing costs are expected to increase. This trend is validated in Figs. 10∼12(e). As the trajectory length becomes larger, the (accumulated) benefit of INS becomes more significant. An interesting observation is that for INS, the random query trajectories have lower server side computation cost and communication cost, but higher client side computation cost. This can be explained by that, for a random query trajectory, the query object has a higher probability to move back and forth in a local region. There is a good chance that when the *k*NN set becomes invalid, we can compute the new *k*NNs from the IS on the client side and hence save the server side costs as well as the communication cost. V*-Diagram and LNVD do not show a similar behavior because they do not have the IS to help compute the new *k*NN sets on the client side.

Varying data object density. We vary the data object density by using different ratios (5% to 10.5%) of the spatial network vertices as data objects. Figs. 10∼12(f) show the results. INS outperforms both V*-Diagram and LNVD again. When the ratio increases, the communication costs of both INS and V*-Diagram increase. This is because the data objects become more dense and the safe regions computed become smaller. More frequent *k*NN set transmission is required as a result. LNVD behaves differently because its search range for building a local Voronoi diagram is determined by the object-vertex ratio, which allows it to compute safe regions based on data object density. The computation costs (server response time and client response time) of all three algorithms decrease. This is because as the data objects become more dense, the algorithms can compute the *k*NN sets by searching in a smaller area of the network, which reduces the computation costs more.

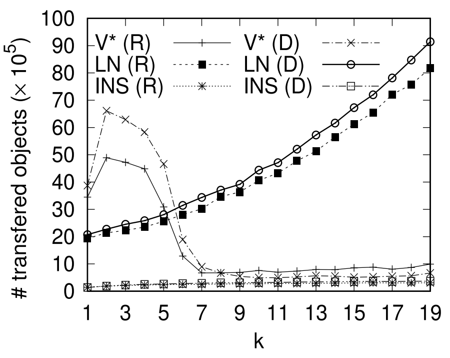
Comparing Metric Spaces As there is no existing method that can apply across all these spaces, we compare the performance



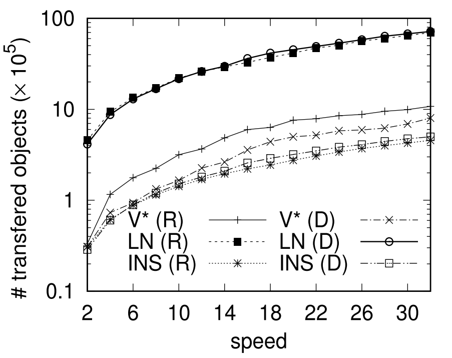
(a) Optimal value of x



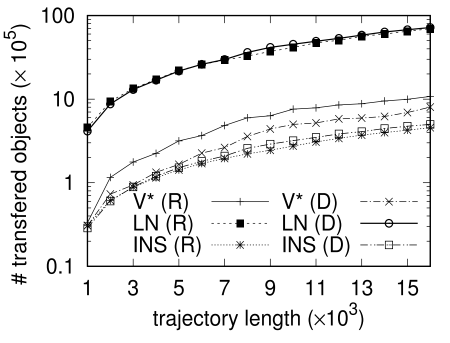
(b) Effect of ρ



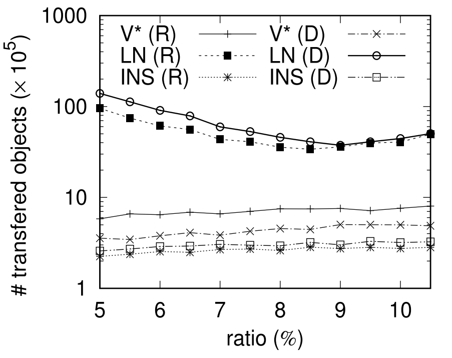
(c) Effect of k



(d) Effect of moving speed

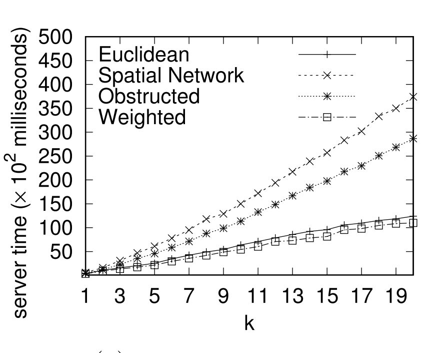


(e) Effect of trajectory length

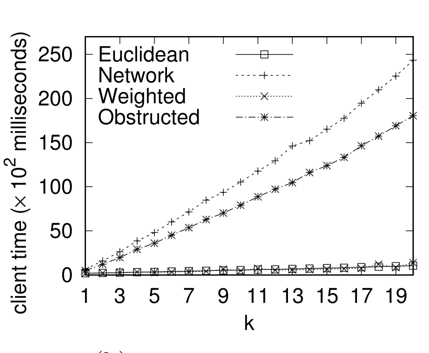


(f) Effect of data object density

Fig. 12. Communication cost.



(a) Server response time



(b) Client response time

Fig. 13. Spaces comparison.

of our algorithm with ourselves. The experiments were run on the USA data set (23, 947, 347 vertices and 58, 333, 344 edges). We run our spatial network algorithm (denoted by **Spatial Network**) on the network consisting all the vertices and edges, run our Euclidean space algorithm (denoted by **Euclidean**) on only the vertices (without considering the underlying road network), run our obstructed version algorithm (denoted by **Obstructed**) considering randomly generated obstacles, and run our weighted version algorithm (denoted by **Weighted**) considering randomly assigned weights on the vertices. We generated obstacles that the number of edges in the visibility network is the same as that in the Spatial Network. The range of weight value is between 0 to 10. Fig. 13 shows the result. We can see that for both server and client response time, the algorithm performance in spatial network and obstructed space are an order of magnitude slower than that in Euclidean space and weighted space. The result confirms the discussion in Section 6 that it requires more time to run our method in spatial networks and obstructed spaces than in Euclidean spaces and weighted spaces, which is due to the extra cost of computing shortest distances between objects in networks. The result also confirms that the time complexity of our algorithm in obstructed spaces is similar to that in spatial networks, and the time complexity of our algorithm in weighted spaces is similar to that in Euclidean spaces.

9. Related work

M*k*NN query in Euclidean setting. In an early study [3], a sampling based approach is used, which has to recompute *k*NN queries at a high frequency to approximate a continuous query, and hence is very expensive.

Recent studies [2,7,8] on the M*k*NN query have adapted the safe region based approach to avoid the sampling cost. The safe region based approach maintains a *k*NN set and an associated "safe region" where the query object can move freely without invalidating this *k*NN set.

The order-*k* Voronoi diagram (*k*VD) [7] and the *Retrieve-Influence-Set* *k*NN algorithm (RIS-*k*NN) [8] are traditional safe region based approaches. However, they need either expensive precomputation or expensive query-time computation. A related data structure, the *VoR-tree* [16], combines the Voronoi diagram and the R-tree. Using the VoR-tree, a *k*NN query can be computed by first finding the top nearest neighbor in the R-tree and then retrieving other nearest neighbors incrementally based on the neighborhood relationship between the objects.

The state-of-the-art M*k*NN algorithm is V*-Diagram [2,12]. We used this algorithm as the baseline algorithm in the experiments and detailed it in Section 2.4. Some other studies [4,20] on M*k*NN queries assume a predefined linear trajectory of the query object. In this case, a safe region is reduced to a line segment on the trajectory.

A comprehensive survey of continuous query processing methods can be found in [21].

M*k*NN query in spatial network setting. Kolahdouzan and Shahabi [13] propose to precompute order-1 Voronoi cells on a spatial network and traverse the neighboring Voronoi cells to identify the *k*NN set at query time. This approach can avoid traversing the whole network for answering a single *k*NN query, but it is not optimized for the M*k*NN query. V*-Diagram has also been extended to the spatial network setting [12]. We described the extension in Section 2.4 as well. Nutanong et al. [19] propose the *Local Network Voronoi Diagram* (LNVD) algorithm which computes a local network Voronoi diagram within a small range around the query object where the range size is determined empirically. We also compared with this algorithm in the experiments.

Huang et al. [22] propose a data model and a definition for the abstract functionality of a spatial network NN search. Papadias et al. [23] utilize the distance in the Euclidean space to facilitate NN search in a spatial network. They are not for M*k*NN processing and are not discussed further. They introduce an architecture that integrates the distance information in the Euclidean space and the spatial network with pragmatic constraints, and generates the search region by expanding from the query object on the network. However, the performance of this method relies heavily on the ratio between the number of data objects and the number of network nodes.

Other related studies. *K*NN queries in metric spaces have been considered. For example, the M-tree [17] is an index for spatial query processing in metric spaces. This index can be used for *k*NN set recomputation for M*k*NN queries in metric spaces. In moving range queries, two concepts named the *guard objects* [24] and *safe exits* [25] have been used for safe region formulation. They guard an "object in query range" relationship, but do not guarantee the nearness rank of the objects. Our safe guarding objects (influential set), on the other hand, guard data objects to be in the *k*NN set. The AP-Tree [26] processes continuous spatial-keyword queries, which considers keyword attributes in addition to spatial coordinates of objects. The AP-Tree technique groups registered queries first, and then processes the grouped queries over streaming data. Fang et al. [27] propose to find *k*NNs between two sets of trajectories, and parallelize their algorithm using MapReduce. Li et al. [28] propose to construct and partition Network Voronoi Diagrams to process continuous NN queries in road networks under data broadcast environments. By balancing the workload between server side and client side, they minimize the data that need to be broadcasted.

Queries over moving objects in spatial networks are studied [29,30], where the data objects are moving and the focus is to minimize the query recomputation while they are moving. In our work we assume that all the data objects are static and only the query object moves. We do not need to consider data object movements in algorithm design and hence their optimization techniques do not apply. On the other hand, the static-object setting allows us to pre-compute order-1 Voronoi cells to boost the query performance. Such pre-computation is infeasible in the previous studies [29,30].

Other types of queries on moving objects have also been studied extensively. These include range queries [31,32], density queries [33], intersection join queries [34], obstructed NN queries [35,36], visible *k*NN queries [37], weighted NN queries [38], destination prediction queries [39], etc. These studies have different problem settings from ours and their solutions are inapplicable.

10. Conclusions

We revisited the M*k*NN query and presented novel algorithms that process the M*k*NN query efficiently in the metric space. The proposed algorithms take advantage of the order-*k* Voronoi cell to the full extent but avoids the high cost of building and validating the order-*k* Voronoi cell. This is achieved by using a concept called the influential neighbor set, which contains a small number of data objects and can be used to validate the current *k*NN set. As a result, we replace the computation of order-*k* Voronoi cells with the computation of influential neighbor sets, which is much more efficient. Our influential neighbor set based algorithms also support data object updates as well as setting *k* at query time since we do not need to precompute any order-*k* Voronoi cell. We conducted extensive experiments using both real and synthetic data sets. The results show that the proposed algorithms outperform the state-of-the-art algorithms on both I/O and computational costs constantly.

This study opens up a few directions to be explored in the future: (i) finding smaller IS would help further reduce the query costs; (ii) applying the concepts of IS and INS to other special metric spaces such as L_1 spaces would further demonstrate the applicability of these concepts; and (iii) extending these concepts to quasi-metric spaces such as directed spatial networks is a challenging problem.

Acknowledgment

This work is supported by the National Key R&D Program of China (2018YFB1003404), the National Nature Science Foundation of China (61872071, 61872070, 61871107), the Fundamental Research Funds for the Central Universities (N171604008, N171605001), the Ministry of Education Joint Foundation for Equipment Pre-Research (6141A020333), and the Australian Research Council (ARC) Discovery Project DP180103332.

References

[1] R. Benetis, C.S. Jensen, G. Karĉiauskas, S. Ŝaltenis, Nearest and reverse nearest neighbor queries for moving objects, VLDBJ 15 (3) (2006) 229–249.
[2] S. Nutanong, R. Zhang, E. Tanin, L. Kulik, The v*-diagram: a query-dependent approach to moving knn queries, PVLDB 1 (1) (2008) 1095–1106.
[3] Z. Song, N. Roussopoulos, K-nearest neighbor search for moving query point, in: SSTD, 2001, pp. 79–96.
[4] Y. Tao, D. Papadias, Q. Shen, Continuous nearest neighbor search, in: VLDB, 2002, pp. 287–298.
[5] G.R. Hjaltason, H. Samet, Ranking in spatial databases, in: SSD, 1995, pp. 83–95.
[6] N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, SIGMOD Rec. 24 (2) (1995) 71–79.
[7] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, Vol. 501, Wiley.com, 2009.
[8] J. Zhang, M. Zhu, D. Papadias, Y. Tao, D.L. Lee, Location-based spatial queries, in: SIGMOD, 2003, pp. 443–454.
[9] L. Kulik, E. Tanin, Incremental rank updates for moving query points, in: GIS, 2006, pp. 251–268.
[10] C. Li, Y. Gu, J. Qi, G. Yu, R. Zhang, W. Yi, Processing moving knn queries using influential neighbor sets, PVLDB 8 (2) (2014) 113–124.
[11] C. Li, Y. Gu, J. Qi, R. Zhang, G. Yu, A safe region based approach to moving knn queries in obstructed space, Knowl. Inf. Syst. (2014).
[12] S. Nutanong, R. Zhang, E. Tanin, L. Kulik, Analysis and evaluation of v*-knn: an efficient algorithm for moving knn queries, VLDBJ 19 (3) (2010) 307–332.
[13] M. Kolahdouzan, C. Shahabi, Voronoi-based k nearest neighbor search for spatial network databases, in: VLDB, 2004, pp. 840–851.
[14] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The r*-tree: an efficient and robust access method for points and rectangles, in: SIGMOD, 1990, pp. 322–331.
[15] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, Springer, 2008.
[16] M. Sharifzadeh, C. Shahabi, Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries, PVLDB 3 (1–2) (2010) 1231–1242.
[17] P. Ciaccia, M. Patella, P. Zezula, M-tree: An efficient access method for similarity search in metric spaces, in: VLDB, 1997, pp. 426–435.
[18] E. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1) (1959) 269–271.
[19] S. Nutanong, E. Tanin, M.E. Ali, L. Kulik, Local network voronoi diagrams, in: GIS, 2010, pp. 109–118.
[20] Y. Tao, D. Papadias, Time-parameterized queries in spatio-temporal databases, in: SIGMOD, 2002, pp. 334–345.
[21] J. Qi, R. Zhang, C.S. Jensen, K. Ramamohanarao, J. HE, Continuous spatial query processing: A survey of safe region based techniques, ACM Comput. Surv. 51 (3) (2018) 64.
[22] X. Huang, C.S. Jensen, S. Šaltenis, The islands approach to nearest neighbor querying in spatial networks, in: SSTD, 2005, pp. 73–90.
[23] D. Papadias, J. Zhang, N. Mamoulis, Y. Tao, Query processing in spatial network databases, in: VLDB, 2003, pp. 802–813.
[24] M.A. Cheema, L. Brankovic, X. Lin, W. Zhang, W. Wang, Multi-guarded safe zone: An effective technique to monitor moving circular range queries, in: ICDE, 2010, pp. 189–200.
[25] D. Yung, M.L. Yiu, E. Lo, A safe-exit approach for efficient network-based moving range queries, Data Knowl. Eng. 72 (2012) 126–147.
[26] X. Wang, Y. Zhang, W. Zhang, X. Lin, W. Wang, Ap-tree: Efficiently support continuous spatial-keyword queries over stream, in: Data Engineering (ICDE), 2015 IEEE 31st International Conference on, IEEE, 2015, pp. 1107–1118.
[27] Y. Fang, R. Cheng, W. Tang, S. Maniu, X. Yang, Scalable algorithms for nearest-neighbor joins on big trajectory data, IEEE Trans. Knowl. Data Eng. 28 (3) (2016) 785–800.
[28] Y. Li, J. Li, L. Shu, Q. Li, G. Li, F. Yang, Searching continuous nearest neighbors in road networks on the air, Inf. Syst. 42 (2014) 177–194.
[29] B. Cao, C. Hou, S. Li, J. Fan, J. Yin, B. Zheng, J. Bao, Simknn: A scalable method for in-memory knn search over moving objects in road networks, IEEE Trans. Knowl. Data Eng. (2018) 322–331.
[30] C. Li, Y. Gu, J. Qi, J. He, Q. Deng, G. Yu, A gpu accelerated update efficient index for knn queries in road networks, in: 2018 IEEE 34th International Conference on Data Engineering (ICDE), IEEE, 2018, pp. 881–892.
[31] M.E. Ali, E. Tanin, R. Zhang, L. Kulik, A motion-aware approach for efficient evaluation of continuous queries on 3D object databases, VLDBJ 19 (5) (2010) 603–632.
[32] R. Zhang, H.V. Jagadish, B.T. Dai, K. Ramamohanarao, Optimized algorithms for predictive range and knn queries on moving objects, Inf. Syst. 35 (8) (2010) 911–932.
[33] C.S. Jensen, D. Lin, B.C. Ooi, R. Zhang, Effective density queries on continuously moving objects, in: ICDE, 2006, pp. 71–82.
[34] R. Zhang, J. Qi, D. Lin, W. Wang, R.C.-W. Wong, A highly optimized algorithm for continuous intersection join queries over moving objects, VLDBJ 21 (4) (2012) 561–586.
[35] Y. Gao, B. Zheng, Continuous obstructed nearest neighbor queries in spatial databases, in: SIGMOD, 2009, pp. 577–590.
[36] C. Li, Y. Gu, F. Li, M. Chen, Moving k-nearest neighbor query over obstructed regions, in: Asia-Pacific Web Conference (APWEB), 2010, pp. 29–35.
[37] Y. Wang, R. Zhang, C. Xu, J. Qi, Y. Gu, G. Yu, Continuous visible k nearest neighbor query on moving objects, Inf. Syst. 44 (2014) 1–21.
[38] C. Li, Y. Gu, G. Yu, F. Li, Wneighbors: a method for finding k nearest neighbors in weighted regions, in: DASFAA, 2011, pp. 134–148.
[39] A.Y. Xue, J. Qi, X. Xie, R. Zhang, J. Huang, Y. Li, Solving the data sparsity problem in destination prediction, VLDBJ 24 (2) (2015) 219–243.