Processing Moving *k*NN Queries Using Influential Neighbor Sets

Chuanwen Li[1], Yu Gu[1], Jianzhong Qi[2], Ge Yu[1], Rui Zhang[2], Wang Yi[1]
[1]College of Information Science and Engineering, Northeastern University, China
{lichuanwen, guyu,yuge,wangyi}@ise.neu.edu.cn
[2]Department of Computing and Information Systems, University of Melbourne
{jianzhong.qi, rui.zhang}@unimelb.edu.au

ABSTRACT

The moving k nearest neighbor query, which computes one's k nearest neighbor set and maintains it while at move, is gaining importance due to the prevalent use of smart mobile devices such as smart phones. Safe region is a popular technique in processing the moving k nearest neighbor query. It is a region where the movement of the query object does not cause the current k nearest neighbor set to change. Processing a moving k nearest neighbor query is a continuing process of checking the validity of the safe region and recomputing it if invalidated. The size of the safe region largely decides the frequency of safe region recomputation and hence query processing efficiency. Existing moving k nearest neighbor algorithms lack efficiency due to either computing small safe regions and have to recompute frequently or computing large safe regions (i.e., an order-k Voronoi cell) with a high cost.

In this paper, we take a third approach. Instead of safe regions, we use a small set of safe guarding objects. We prove that, as long as the the current k nearest neighbors are closer to the query object than the safe guarding objects, the current k nearest neighbors stay valid and no recomputation is required. This way, we 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. This means that the recomputation frequency of our method is also minimized. We conduct extensive experiments comparing our method with the state-of-the-art method on both real and synthetic data sets. The results confirm the superiority of our method.

1. INTRODUCTION

Smart mobile devices and *location-based services (LBS)* have become prevalent. On-going efforts have been made to improve the use experience of mobile LBS through improving moving query processing efficiency. In this paper we revisit a major type of moving query, the *moving k nearest neighbor (M*k*NN)* query [4, 16, 21, 23]. *Given a moving query object and a set of static data objects, the M*k*NN query reports the k nearest neighbors of the query object continuously while it is moving*. For example, an MkNN query can be used to report the 3 nearest gas stations continuously while one drives on highway, or the 5 nearest points of interest (POI) continuously while a tourist is walking around the city.

Computing the kNN from a static data set for a static query object has been studied extensively (e.g., [7, 19]). However, existing techniques for the static kNN query is not directly applicable for the MkNN query. This is because, when the query object is moving, keeping its kNN set up-to-date requires constant kNN recomputation if a static kNN query algorithm were used, which is too expensive to complete in time.

Recent studies on the MkNN query have adapted the *safe region based* approach [16, 18, 25]. The idea of safe region is based on that objects at nearby positions often share the same kNN set. There may be a region where all points have the same kNN set. Inside such a region, an object is "safe" to move freely without causing its kNN set to change. Thus, such a region is called a safe region. Using the safe region significantly reduces the MkNN query processing cost because the kNN set will only need 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 kNN 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 object location updates.

Existing methods either fall short in construction overhead or validation overhead. Specifically, earlier studies [12, 18, 25] use *Voronoi cells* [18] as the safe regions, which have high construction overhead. A more recent study [16] uses relaxed safe regions to reduce the construction overhead, but they have to recompute the safe regions more frequently and have higher validation overhead.

A Voronoi cell based safe region is essentially an *order-k Voronoi cell*. Given a kNN set, its order-k Voronoi cell is defined as a region where the kNN set stays valid [18]. Figure 1 (c) illustrates the Voronoi cells on $O = \{p_1, \ldots, p_{11}\}$, where the dashed lines indicate boundaries of order-1 Voronoi cells, respectively. An object that moves inside the order-1 Voronoi cell of p_3 will always view p_3 as its nearest neighbor. Similarly, an object that moves inside the order-2 Voronoi cell of $\{p_4, p_7\}$ (denoted by the cross-lined region) will always view $\{p_4, p_7\}$ as its 2NN set. Here, an order-2 Voronoi cell is computed by combining two order-1 Voronoi diagrams. To compute the order-2 Voronoi cell of $\{p_4, p_7\}$, we first compute two order-1 Voronoi diagrams, one ignoring p_4 and the other p_7, as shown in Figures 1 (a) and (b), 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 becomes large, the cost of computing order-k Voronoi cells becomes prohibitive. Pre-

Proceedings of the VLDB Endowment, Vol. 8, No. 2

computing the order-k Voronoi cells [12, 25] is also unpractical due to the rapid increase in the number of order-k Voronoi cells as k increases. Plus, precomputing the order-k Voronoi cells loses the flexibility of setting k at query time, which is a significant disadvantage [18].

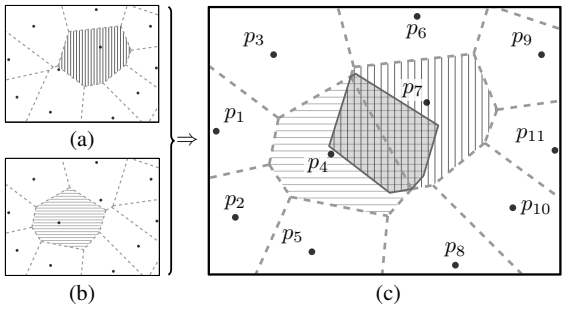


Figure 1: Voronoi cells (order-1 and order-2)

The state-of-the-art MkNN algorithm, V*-Diagram [16], avoids computing the exact order-k Voronoi cells and hence reduces the construction overhead. It uses a safe region combined from *safe region w.r.t. a data object* and *fixed-rank region*, which are simpler to compute but are also less strict than the order-k Voronoi cells. Thus, it has to recompute the safe regions more frequently and has higher validation overhead.

In this paper, we propose a novel approach that validates kNN sets as strict as the order-k Voronoi cells, but do not have to compute the exact order-k Voronoi cells (or any safe region at all). Our main idea is to use *safe guarding objects* instead of safe regions. The intuition is that, since the query object moves continuously, when the kNN set changes, some data object near the current kNN objects must become one of the new kNNs. We call this type of data objects (i.e., data objects near the current kNN objects) the safe guarding objects. As long as no safe guarding object becomes a kNN, the current kNN 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. Thus, they guarantee minimum kNN set recomputation and hence minimum communication between the query client and the query processor, which is critical in LBS. For efficient computation we identify a special set of safe guarding objects and call it the *influential neighbor set*. This set guarantees the validity of the kNN set, while can be computed and validated in time linear to k. Therefore, it can reduce both the construction overhead and validation overhead at the same time. In addition, we do not need to precompute the influential neighbor sets, and hence we allow setting k at query time.

We further propose an MkNN algorithm based on the influential neighbor set and make the following contributions:

- We analyze the existing Voronoi cell based techniques for the MkNN query thoroughly 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 kNN sets.
- We propose the *influential neighbor set*, which is a special type of influential set that can be retrieved and validated efficiently, and further obtain an algorithm that overcomes the drawbacks of the existing techniques. Our algorithm has the following advantages:
  - **Efficient kNN set validation.** We just need a single scan on the kNN set and the small influential neighbor set to validate the kNN set.
  - **Efficient influential set computation.** We compute the influential set when it becomes invalid. This is much less costly than computing the order-k Voronoi cell based safe regions, for that the influential set can be computed in time linear to k (detailed in Section 3.2).
  - **Data update support.** Our algorithm supports data object updates on-the-fly. Insertion and deletion can be processed with both space and time complexity at $\mathcal{O}(k \log k)$.
- We perform cost analysis and extensive experiments, and the results confirm the superiority of our algorithm over the state-of-the-art algorithm.

The remainder of the paper is organised as follows. We review related studies in Section 2 and present the preliminaries in Section 3. The concept of influential set and influential neighbor set are introduced in Section 4 and Section 5, respectively. Our query processing algorithm is described in Section 6. We analyse the costs of our algorithm in Section 7, report the experimental results in Section 8 and conclude the paper in Section 9.

2. RELATED WORK

As a major type of spatial query over moving objects, the MkNN queries have attracted a large body of studies [8, 9, 16, 21, 23].

In an early study [21], a sampling based approach is used. This approach samples objects from the query object's trajectory and processes a kNN query for each sampled position. To approximate a continuous query, the sampling based approach has to recompute kNN queries at a high frequency, which is very expensive.

Due to the high cost of the sampling based approach, recent studies [16, 18, 25] on the MkNN query have adapted the safe region based approach. The safe region based approach maintains a kNN set and an associated "safe region" where the query object can move freely without invalidating this kNN set. The query processor only needs to process a kNN query when the query object moves out of the safe region. Thus, both the computation cost on the query processor and the communication cost between the query object and the query processor are reduced. For each kNN set, there is an infinite number of safe regions which can guarantee the validity of the kNN set. The largest among these safe regions is the order-k Voronoi cell built on the kNN set.

The order-k Voronoi diagram (kVD) [18] approach is an example of using the order-k Voronoi cells as the safe regions. The problem of this approach is that it has to precompute too many order-k Voronoi cells, which are expensive to compute and store. The *Retrieve-Influence-Set* kNN algorithm (RIS-kNN) [25] avoids the high precomputation cost of kVD by computing an order-k Voronoi cell locally on-the-fly. Every time the kNN set is to be updated, it requires issuing a number of (six on average) expensive time-parameterized kNN queries to rebuild the safe region. A related data structure, the *VoR-tree* [20], combines the Voronoi diagram and the R-tree. Using the VoR-tree, a kNN query can be computed by first finding the 1 nearest neighbor in the R-tree and then retrieving other nearest neighbors incrementally based on the neighborhood relationship between the objects. This technique reduces the cost of processing one kNN query. However, it does not help improve the continuous processing efficiency of the MkNN query.

The state-of-the-art MkNN algorithm is V*-Diagram [16, 17]. We use this algorithm as the baseline algorithm in the experiments and will detail it in Section 3.2.

Some other studies [22, 23] on the MkNN query assume a predefined linear trajectory of the query object. In this case, the safe region is reduced to a line segment on the predefined trajectory, and can be determined by the bisectors between the query object

and its nearby data objects with low costs. In recent years, other types of queries on moving objects have also been studied extensively. These include the range queries [1, 26], the kNN queries with two predicates [2], the density queries [10], the intersection join queries [27, 28], the obstructed NN queries [5, 13], the visible NN queries [6], the weighted NN queries [14] and the destination prediction queries [24], etc. These studies have different problem settings from ours and their solutions are inapplicable.

3. PRELIMINARIES

We first present a definition to the MkNN query and some basic concepts. We summarize the frequently used symbols in Table 1.

Table 1: Frequently Used Symbols

Symbol	Meaning
k	The number of queried nearest neighbors.
ρ	The prefetch ratio. ($\rho \geq 1$)
O	The set of data objects.
$d(p_1, p_2)$	The distance between objects p_1 and p_2.
$b(p_1, p_2)$	The bisector between objects p_1 and p_2.
$NN_k(q)$	The k nearest neighbors of q.
$I(O')$	The influential neighbor set of a set O'.
$N_{O'}(p_i)$	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'.
$V^k(O')$	The order-k Voronoi cell of object set O' in the Voronoi diagram with respect to object set O.
$V^k(O') \parallel V^k(O'')$	The two Voronoi cells share an edge.
$A \prec_q B$	Any object in set A is nearer to q than any object in set B.
$A \circlearrowleft B$	Set A is an influential set of set B.

We consider 2-dimensional point data in the Euclidean space. Given a moving query object q, a set of static data objects O, where each object can be seen as a point, and a query parameter k, the MkNN query returns the kNN set (from O) of q continuously (i.e., at every timestamp). Let $d(p_1, p_2)$ be a function that returns the Euclidean distance between two objects p_1 and p_2. We have the following formal query definition.

DEFINITION 1. *(MkNN) Given a moving query object q, a set of static data objects $O = \{p_1, p_2, ..., p_n\}$, and a query parameter k ($n \geq k$), at every timestamp, the* moving k nearest neighbor (MkNN) *query returns a subset $O' \subseteq O(|O'| = k)$: $\forall p' \in O'$ $\forall p \in O \backslash O'$, $d(p', q) \leq d(p, q)$.*

The Voronoi diagram gives a basic safe region for processing the MkNN query.

3.1 Voronoi Diagram

DEFINITION 2. *(Voronoi Diagram) The* Voronoi diagram *[18] of a set of n data objects O, is a subdivision of the plane into n* Voronoi cells*, where each Voronoi cell corresponds to a data object p_i in O and it has the following property:*

Let $V_O(p_i)$ be the Voronoi cell of p_i. Then for any point o within $V_O(p_i)$, p_i is the nearest neighbor of o from the set O, i.e., $\forall p_j \in O \backslash p_i$, $d(o, p_i) \leq d(o, p_j)$.

Figure 1 (c) gives an example, where the dots denote the data objects and the dashed lines denote the (order-1) Voronoi cells.

The Voronoi diagram is computed using the bisectors between the data objects in O. However, not every bisector between the data objects is needed. Only those between adjacent data objects will be included in the Voronoi diagram. This is because the bisector between p_i and a far away object p_j will be shadowed by the Voronoi cells of the objects between p_i and p_j anyway. For example, in Figure 1 (c), the bisector between p_4 and p_9 is shadowed by $V_O(p_7)$.

Two adjacent data objects p_i and p_j whose bisector is included in the Voronoi diagram are called the *Voronoi neighbors*, i.e., p_i and p_j share an edge of their Voronoi cells, denoted by $V_O(p_i) \parallel V_O(p_j)$. Okabe et al. [18] prove that, as stated in Lemma 1, if there is a circle where p_i and p_j are on its boundary, while this circle encloses no other data objects, then p_i and p_j are Voronoi neighbors. For example, in Figure 2, the Voronoi cells of p_{10} and p_{11} share an edge, and hence p_{10} and p_{11} are Voronoi neighbors. We can see that there is a circle o_1 where p_{10} and p_{11} are on its boundary, and this circle does not enclose any other data objects.

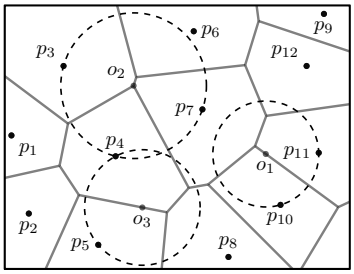


Figure 2: Example of Lemma 1

LEMMA 1. *Two data objects p_i and p_j are Voronoi neighbors if and only if there exists a circle where p_i and p_j are on its boundary, and no other data object is enclosed by the circle. [18]*

The Voronoi cells defined above is also called order-1 Voronoi cells. By definition they can be used as safe regions for 1NN (i.e., $k = 1$) queries. When $k > 1$, we need Voronoi diagrams and Voronoi cells of higher order, which are defined as follows.

DEFINITION 3. *(Order-k Voronoi Diagram) The* order-k Voronoi diagram *[18] of a set of data objects O, is a subdivision of the plane into* order-k Voronoi cells*, where each order-k Voronoi cell corresponds to a subset O' of k data objects in O and it has the following property:*

Let $V^k(O')$ be the order-k Voronoi cell of O'. Then for any point o within $V^k(O')$, O' is the kNN set of o, i.e.,

$$\forall p_i \in O' \ \forall p_j \in O \backslash O', \ d(o, p_i) \leq d(o, p_j). \quad (1)$$

In Figure 1 (c), the cross-lined region denotes the order-2 Voronoi cell of $\{p_4, p_7\}$, $V^2(p_4, p_7)$. Any point in this cell will view $\{p_4, p_7\}$ 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 $V^k(O')$ can be computed from order-1 Voronoi cells as follows. For a data object $p \in O'$, we compute an order-1 Voronoi cell $V_{O \backslash O' \cup \{p\}}(p)$ using the data points in $O \backslash O' \cup \{p\}$. This will give a region where objects view p as the nearest neighbor (ignoring other objects in O'). We do this for every data object in O' and obtain k order-1 Voronoi cells where objects view the k objects in O' as the nearest neighbors, respectively. The intersection of these k order-1 Voronoi cells is the order-k Voronoi cell $V^k(O')$. Formally,

DEFINITION 4. *Given a set of data objects O and a subset O' of O where $|O'| = k$, the order-k Voronoi cell of O' is computed as:*

$$V^k(O') = \bigcap_{p \in O'} V_{(O \setminus O' \cup \{p\})}(p). \quad (2)$$

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

Discussion. By definition, the order-k Voronoi cell of O' gives the largest possible safe region for O'. The kNN set of a query object is O' if and only if the query object stays in the order-k Voronoi cell of O'. 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 can be seen from Definition 4, computing an order-k Voronoi cell requires computing k order-1 Voronoi cells, where each Voronoi cell requires computing the bisectors between the data objects in different object sets. *This is prohibitive considering the number of possible order-k Voronoi cells, and is also inflexible with regard to changes of k or the data set O, which is a significant disadvantage [18].*

3.2 V*-Diagram

V*-Diagram [16] is the state-of-the-art solution for MkNN queries. It uses a safe region called the *Integrated Safe Region* (ISR), which is the intersection of two types of regions: (i) the *safe region w.r.t. the k^{th} nearest data object*, and (ii) the *fixed-rank region* of the $k+x$ nearest data objects, where x is the number of *auxiliary data objects* maintained to reduce safe region recomputation.

The V*-Diagram MkNN algorithm computes the integrated safe region as follows. It first computes the query object's $(k+x)$ nearest data objects. Let q_c be the current position of the query object q and z be its $(k+x)^{th}$ nearest data object. Then a *known region* is computed as a disk centered at q_c with the radius $d(q_c, z)$.

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

$$\omega(q_c, p, d(q_c, z)) = \{q' | d(q', p) \leq d(q', p')\}. \quad (3)$$

According to triangle inequality, $d(q_c, q') + d(q', p') \geq d(q_c, p')$. The definition of $\omega(q_c, p, d(q_c, z))$ can be tightened by replacing $d(q', p')$ with $d(q_c, p') - d(q_c, q')$:

$$\omega(q_c, p, d(q_c, z)) = \{q' | d(q', p) \leq d(q_c, p') - d(q_c, q')\}. \quad (4)$$

Object p' is outside the know region, i.e., $d(q_c, p') \geq d(q_c, z)$. Thus, the equation is further simplified to be:

$$\begin{aligned} \omega(q_c, p, d(q_c, z)) &= \{q' | d(q', p) \leq d(q_c, z) - d(q_c, q')\} \\ &= \{q' | d(q', p) + d(q_c, q') \leq d(q_c, z)\}. \end{aligned} \quad (5)$$

This equation shows that the safe region w.r.t. p is essentially an ellipse in the Euclidean space where q_c and p are its two foci and $d(q_c, z)$ is its major axis length.

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' outside the known region cannot be closer to q than any of those k data objects.

Step 2. The V*-Diagram algorithm further computes a region called the *fixed-rank region* where the order of distances 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, $\eta\langle p_1, p_2, ..., p_{k+x}\rangle$, 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} ($i \in [1 \cdots k+x-1]$):

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 [16]:

$$\Omega(q_c, L_{k+x}) = \eta(L_{k+x}) \cap \omega(q_c, p_k, d(q_c, z)). \quad (6)$$

Figure 3 shows an example where $k = 2$ and $x = 2$. When the query object q is at the initial location q_c, a 4NN search retrieves the 4 nearest data objects $\langle p_1, p_3, p_2, p_6\rangle$. The ellipses filled with horizontal lines and vertical lines denote $\omega(q_c, p_1, d(q_c, p_6))$ and $\omega(q_c, p_3, d(q_c, p_6))$, respectively. Then as long as q remains in the grey region $\eta\langle p_1, p_3, p_2, p_6\rangle \cap \omega(q_c, p_3, d(q_c, p_6))$, the 2NN of q will not change.

$$\eta\langle p_1, p_2, ..., p_{k+x}\rangle = \bigcap_{i=1}^{k+x-1} H(p_i, p_{i+1}). \quad (7)$$

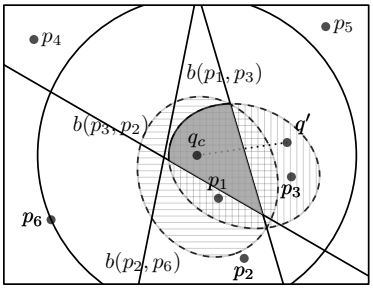


Figure 3: Integrated safe region ($k = 2$, $x = 2$)

Discussion. V*-Diagram still uses the distance difference between the objects in the current kNN set of q and the objects that are not to define the safe region. It uses the safe region w.r.t. the k^{th} NN to replace the Voronoi cells, which reduces the safe region computation complexity and saves time. However, the safe region obtained is not as large as the order-k Voronoi cell. This means that the safe regions and the kNN set have to be recomputed more frequently, and will bring in unnecessary query processing overhead. It motivates us to find a solution to process the MkNN query, such that we achieve kNN recomputation frequency as low as that when the order-k Voronoi cells are used as the safe regions.

4. INFLUENTIAL SET

The key idea of our MkNN query processing approach is to use a set of *safe guarding objects*. As long as the query object is closer to the current kNNs than the safe guarding objects, it is guaranteed that the current kNN set is still valid, and therefore, we do not need to recompute the kNN set.

We call a set of safe guarding objects an *influential set*, in the sense that they are *influential* in determining whether a kNN set is valid. Formally, an influential set is defined as follows.

DEFINITION 5. *(Influential Set, IS) Given a set of data objects O, a query object q and a* k*NN set $O' \subset O$, we call a set $S \subseteq O \setminus O'$ an* influential set *of O', denoted by $S \circlearrowleft O'$, if O' stays as the* k*NN*

set of q as long as the objects in O' are closer to q than the objects in S are, i.e.,

$$O' = NN_k(q) \iff O' \prec_q S. \quad (8)$$

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.

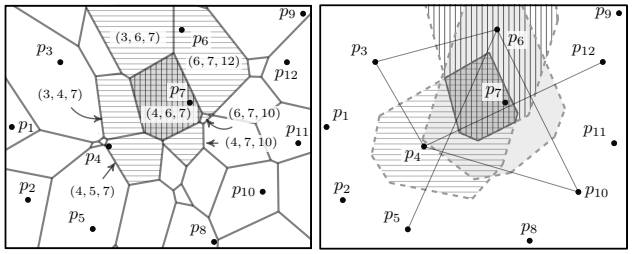
Naively, $O \backslash O'$ is an IS of O', since by definition the kNN set O' is closer to q than $O \backslash O'$. However, using this set to validate the kNN set requires a scan on all objects in O, which means that the solution has fallen back to the naive scan method and is inefficient.

We aim to find the *minimal influential set*, which is the smallest influential set that can guarantee the validity of the current kNN set. We observe that, since the query object q moves continuously, when the current kNN set O' becomes invalid, q must have left the order-k Voronoi cell of the current kNN set, $V^k(O')$, and entered a neighboring order-k Voronoi cell denoted by $V^k(O'')$, where O'' denote the corresponding objects (the new kNN set). We combine the objects corresponding to all neighboring order-k Voronoi cells, excluding those already in O', Then by definition at least one of these objects must become a new kNN when O' becomes invalid. These objects form the minimal influential set defined as follows.

DEFINITION 6. *(Minimal Influential Set, MIS) Given a* k*NN set O', the* minimal influential set *(MIS) of O' is*

$$MIS(O') = \Big(\bigcup_{V^k(O') \parallel V^k(O'')} O'' \Big) \backslash O'. \quad (9)$$

Here, $V^k(O'') \parallel V^k(O')$ means that the two order-k Voronoi cells are Voronoi neighbors, i.e., they share an edge. Figure 4 (a) gives an example, where $O' = \{p_4, p_6, p_7\}$ and the cross-lined region denotes $V^3(O')$. There are five neighboring order-3 Voronoi cells, as denoted by the horizontal-lined regions. The triples associated to the neighboring cells denote the corresponding objects, e.g., $(3,4,7)$ is associated to $V^3_O(p_3, p_4, p_7)$. The union of the triples excluding O' is $3, 5, 10, 12$. Therefore, the MIS of O', $MIS(O') = \{p_3, p_5, p_{10}, p_{12}\}$.



(a) $MIS(O')$ by neighboring cells (b) $MIS(O')$ by bisectors

Figure 4: The minimal influential set (MIS) of $O' = \{p_4, p_6, p_7\}$

To prove that MIS is minimal, we need the following two lemmas. The first lemma suggests that for two neighboring order-k Voronoi cells, their corresponding sets of data objects differ by only one object, e.g., in Figure 4 (a), $V^3_O(p_4, p_6, p_7)$ and a neighboring order-k Voronoi cell $V^3_O(p_3, p_4, p_7)$ share 2 data objects p_4 and p_7 and differ in that the former has p_6 while the latter p_3.

LEMMA 2. *If $V^k(O') \parallel V^k(O'')$, then $|O' \backslash O''| = 1$.*

PROOF. We prove by contradiction. Suppose $|O' \backslash O''| > 1$. Then we can find $\{p_a, p_b\} \subseteq O' \backslash O''$ and $\{p_c, p_d\} \subseteq O'' \backslash O'$.

Let $H(p_m, p_n)$ be the half plane on p_m's side when the space is divided by the bisector $b(p_m, p_n)$ of two objects p_m and p_n. Then,

$$V^k(O') \subset H(p_a, p_c) \cap H(p_a, p_d) \cap H(p_b, p_c) \cap H(p_b, p_d)$$
$$V^k(O'') \subset H(p_c, p_a) \cap H(p_c, p_b) \cap H(p_d, p_a) \cap H(p_d, p_b).$$

We also have $V^k(O') \parallel V^k(O'')$. Thus, the four bisectors $b(p_a, p_c)$, $b(p_a, p_d)$, $b(p_b, p_c)$ and $b(p_b, p_d)$ must overlap with each other, which means that p_a and p_b must overlap with each other, and that p_c and p_d must overlap with each other. Therefore, O' and O'' differ by only one object and $|O' \backslash O''| = 1$. □

The second lemma suggests that each data object in $MIS(O')$ contributes at least one edge of $V^k(O')$.

LEMMA 3. *$MIS(O')$ contains and only contains the data objects that each contributes at least one bisector to form $V^k(O')$.*

PROOF. By definition, the boundary of the order-k Voronoi cell $V^k(O')$ consists of a series of bisectors between a data object in O' and a data object in $O \backslash O'$. These bisectors are shared by $V^k(O')$ and its neighboring order-k Voronoi cells. The bisector shared by $V^k(O')$ and a neighboring order-k Voronoi cell $V^k(O'')$ must be defined by an object $p' \in O' \backslash O''$ and an object $p'' \in O'' \backslash O'$. Since the corresponding object sets of two neighboring order-k Voronoi cells differ by only one object, we know that $O'' \backslash O'$ contains and only contains the object that contribute to a bisector to form $V^k(O')$. $MIS(O')$ is the union of $O'' \backslash O'$ for all O''s. Therefore, $MIS(O')$ contains and only contains the data objects that each contributes at least one bisector to form $V^k(O')$. □

Figure 4 (b) illustrates the lemma. The horizontal-lined, vertical-lined and light gray regions denote $V_{O \backslash O' \cup \{p_4\}}(p_4)$, $V_{O \backslash O' \cup \{p_6\}}(p_6)$ and $V_{O \backslash O' \cup \{p_7\}}(p_7)$ respectively. Their intersection, enclosed by the solid line boundary, is $V^3(O')$. The boundary of $V^3(O')$ is comprised of $b(p_4, p_3)$, $b(p_4, p_{12})$, $b(p_4, p_{10})$, $b(p_6, p_3)$, $b(p_6, p_5)$ and $b(p_6, p_{10})$. The objects which contribute to these bisectors are paired up and linked together. We can see that these objects (excluding those in O') are $\{p_3, p_5, p_{10}, p_{12}\} = MIS(O')$.

Now we can prove that MIS is minimal by proving that any IS must contain MIS.

THEOREM 1. *Given a* k*NN set O' and a set $S \subseteq O \backslash O'$,*

$$S \circlearrowleft O' \iff S \supseteq MIS(O'). \quad (10)$$

PROOF. (i) $S \circlearrowleft O' \Rightarrow S \supseteq MIS(O')$: Suppose $S \not\supseteq MIS(O')$, which means that there exists an object $p \in MIS(O')$, $p \notin S$. Let the query object q be in $V^{k+1}(O' \cup \{p\}) \backslash V^k(O')$, we have $O' \prec_q S$ and $O' \neq NN_k(q)$, thus, S is not an IS of O'. Here, $V^{k+1}(O' \cup \{p\})$ is not empty because an edge of $V^k(O')$ is the bisector between p and some data object in O', and this edge must be in $V^{k+1}(O' \cup \{p\})$.

(ii) $S \supseteq MIS(O') \Rightarrow S \circlearrowleft O'$: Given $S \supseteq MIS(O')$, then (a) when $O' = NN_k(q)$, by the definition of kNN query we have $O' \prec_q S$; (b) when $O' \prec_q S$, since S contains all data objects that contributes a bisector to form the boundary of $V^k(O')$, q must be in $V^k(O')$. □

Discussion. The idea of influential sets essentially replaces the safe regions with safe guarding objects in processing MkNN queries. Similar to computing the strict safe region (i.e., an order-k Voronoi cell), computing the MIS is an expensive operation[1]. To reduce the computational cost, we compute an influential set that is slightly larger than the MIS while can be computed much more efficiently instead. By definition of the influential set, this will not affect the strictness in validating the kNN sets. This is an important advantage of our influential set based method over the safe region based methods as an approximated safe region will not be as strict.

[1]Constructing an order-k Voronoi diagram, which contains $\mathcal{O}(k(n-k))$ cells [18], needs $\mathcal{O}((n + \min\{k(n-k), (n-k)^2\}) \log n)$ time [15].

Various influential sets may be used to approximate the MIS, which may bring a series of new studies on the MkNN query. In this study we propose the *influential neighbor set*, a type of IS that can be constructed and validated efficiently.

5. INFLUENTIAL NEIGHBOR SET

The influential set we use is the *influential neighbor set (INS)*, which is defined based on the Voronoi neighbors.

DEFINITION 7. *(Voronoi neighbor set) Given a Voronoi diagram on data object set O, we call an object p_j a* Voronoi neighbor *of another object p_i if the two Voronoi cells of the two objects share an edge, i.e., $V(p_i) \parallel V(p_j)$. We call $O' \subset O$ that contains all Voronoi neighbors of p_i the* Voronoi neighbor set *of p_i, denoted by $N_O(p_i)$.*

For an order-1 Voronoi diagram, the Voronoi neighbor sets can be precomputed and stored with little overhead [20]. The influential neighbor set is the union of the order-1 Voronoi neighbor sets of all current kNNs. In this section, a Voronoi neighbor refers to an order-1 Voronoi neighbor unless specified otherwise.

DEFINITION 8. *(Influential neighbor set, INS) Given a* k*NN set O', an object p is an* influential neighbor *of O' if p is not in O' while it is a Voronoi neighbor of an object p' in O', i.e., for $p \notin O', \exists p' \in O' : V(p) \parallel V(p')$. We call the set of all influential neighbors of O' the* influential neighbor set (INS) *of O', denoted by $I(O')$,*

$$I(O') = (\bigcup_{p' \in O'} N_O(p')) \backslash O'. \tag{11}$$

Next we prove that an INS is an IS. We first rewrite the definition of the INS such that it is similar to the definition of an order-k Voronoi cell as shown in Equation 2 based on the following lemma.

LEMMA 4. *If a data object p is on the boundary of a circle C and this circle encloses some other data objects, let this set of other data objects be N. Then there exists a data object in N that is one of p's Voronoi neighbors.*

PROOF. We use Figure 5 (a) to illustrate the proof of the lemma. Let o be the center of C (the larger circle in the figure). We first prove that, for any data object p' in N, the bisector $b(p,p')$ intersects the line segment between o and p, denoted by $\overline{op}$. Let the line segment between p and p' be $\overline{pp'}$. We extend $\overline{pp'}$ such that it intersects C at p''. Then $\overline{pp''}$ is a chord of C. By property of a chord, the bisector $b(p,p'')$ passes o. Since p' is nearer to p than p'' is, $b(p,p'')$ intersects $\overline{op}$. We call this the intersection point of p'.

Let p_n be the data object whose intersection point i_n is the nearest to p among all the intersection points of the data objects in N. We prove that p_n is a Voronoi neighbor of p. We draw a circle with i_n as the center and $d(p,i_n)$ as the radius (the smaller circle in the figure). Then this circle satisfies Lemma 1 for p_n and p. In particular, (i) p and p_n are on the boundary of the circle; (ii) any other object in N will not be enclosed by the circle, as otherwise its intersection point would be nearer to p than i_n is; (iii) any object not in N will not be enclosed by this circle as this circle is inside C while an object not in N is not enclosed by C. Therefore, p_n is an object in N that is a Voronoi neighbor of p. □

This lemma states that, if a data object p is on the boundary of a circle, and this circle encloses some other objects, then at least one of these objects must be a Voronoi neighbor of p. For example, in Figure 5 (b), p_{12} is on the boundary of a circle (the largest circle C_2). This circle also encloses $\{p_6, p_7\}$. We can see from the figure that p_6 and p_7 are both Voronoi neighbors of p_{12}.

Now we rewrite the definition of INS.

LEMMA 5. *Given a data set O and a subset O', we have:*

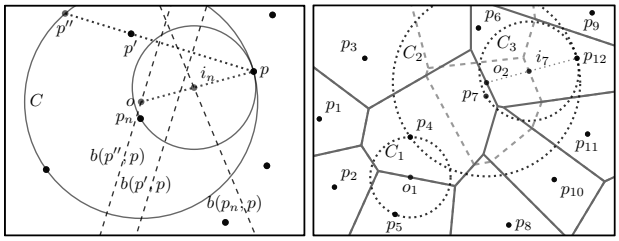
$$I(O') = (\bigcup_{p' \in O'} N_O(p')) \backslash O' = \bigcup_{p' \in O'} N_{O \backslash O' \cup \{p'\}}(p'). \tag{12}$$

PROOF. (i) $(\bigcup_{p' \in O'} N_O(p')) \backslash O' \subseteq \bigcup_{p' \in O'} N_{O \backslash O' \cup \{p'\}}(p')$. An object $p \in (\bigcup_{p' \in O'} N_O(p')) \backslash O'$ indicates that, in the Voronoi diagram of O, there is an object $p' \in O'$ where p is a Voronoi neighbor of p'. From Lemma 1 we know that there is a circle C having p and p' on its boundary while enclosing no other data objects from O. Since $(O \backslash O') \cup \{p'\} \subset O$, we know that in the Voronoi diagram of $(O \backslash O') \cup \{p'\}$, the circle C will still have p and p' on its boundary while enclosing no other data objects. Thus, p is still a Voronoi neighbor of p', i.e., $p \in \bigcup_{\{p'\} \in O'} N_{O \backslash O' \cup \{p'\}}(p')$.

(ii) $\bigcup_{p' \in O'} N_{O \backslash O' \cup \{p'\}}(p') \subseteq (\bigcup_{p' \in O'} N_O(p')) \backslash O'$. Let p and p' be Voronoi neighbors in the Voronoi diagram of $O \backslash O' \cup \{p'\}$, $p' \in O'$. According to Lemma 1, there is a circle C having p and p' on its boundary while enclosing no other objects from $O \backslash O' \cup \{p'\}$. In the Voronoi diagram of O, there are two cases:

(a) The circle C encloses no other data objects. Then p is also a Voronoi neighbor of p' in the Voronoi diagram of O, i.e., $p \in (\bigcup_{p' \in O'} N_O(p')) \backslash O'$.

(b) The circle C encloses other data objects. Then the data objects enclosed in the circle must be those in O', since only data objects in O' have been introduced to the Voronoi diagram. By Lemma 4, p must be a Voronoi neighbor of anther data object in O'. Thus, $p \in (\bigcup_{p' \in O'} N_O(p')) \backslash O'$. □



(a) Lemma 4 (b) Lemma 5

Figure 5: INS redefinition

Figure 5 (b) illustrates Lemma 5, where $O = \{p_1, \ldots, p_{12}\}$ and $O' = \{p_4, p_6, p_7\}$. We take p_5 and p_{12} as an example to show how data objects in $\bigcup_{p' \in O'} N_{O \backslash O' \cup \{p'\}}(p')$ are also in $(\bigcup_{p' \in O'} N_O(p')) \backslash O'$, where $p' \in O'$. These two objects both belong to $N_{(O \backslash O') \cup \{p_4\}}(p_4)$, and hence we can draw two circles on (p_4, p_5) and (p_4, p_{12}), respectively. These two circles do not contain any other data objects in $(O \backslash O') \cup \{p_4\}$, as shown by the two dotted circles C_1 and C_2 centered at o_1 and o_2, respectively.

When considering the neighbors in the Voronoi diagram with regard to O, C_1 remains empty, and hence p_5 will still be in $N_O(p_4) \backslash O'$; C_2 now becomes non-empty since p_6 and p_7 are also used in the Voronoi diagram construction and are enclosed by the circle. However, by Lemma 4 we know that a circle C_3 inside C_2 with i_7 as its center and $d(p_{12}, i_7)$ as its radius has only p_{12} and p_7 on its boundary, and encloses no other objects. Therefore, p_{12} belongs to $N_O(p_7) \backslash O'$, and hence p_{12} is still in $(\bigcup_{p' \in O'} N_O(p')) \backslash O'$.

Comparing Equation 2 with Equation 12, we can see that an INS and the corresponding order-k Voronoi cell are defined based on the same set of data objects. Next we formalize the computation of the order-k Voronoi cell of O' from the INS of O'.

LEMMA 6. *Given a set of data objects O and a* k*NN set O', we have that each edge of $V^k(O')$ is a segment of the bisector $b(p,p')$ of two data objects p and p', where $p \in I(O')$ and $p' \in O'$.*

PROOF. By definition, $V^k(O')$ is the intersection of the (order-1) Voronoi cells of each data object p' in O' in the Voronoi diagram of $O \backslash O' \cup \{p'\}$. Since the edges of these cells are segments of the bisectors of the data objects in O' and their Voronoi neighbors, the order-k Voronoi cell of O' also consists of these bisectors. Therefore, each edge of the order-k Voronoi cell of O' is the bisector of an object p' in O' and an object in $N_{(O \backslash O') \cup \{p'\}}(p')$, which according to Lemma 5, belongs to $I(O')$. □

The above lemmas lead to the fact that an INS is an IS, as formalized by the following theorem.

THEOREM 2. *Given a* k*NN set* O', $I(O') \circlearrowleft O'$.

PROOF. According to Lemma 3 and Lemma 6, we have $I(O') \supseteq MIS(O')$ and hence $I(O')$ is an IS. □

Next, we detail our M*k*NN algorithm based on INS.

6. QUERY PROCESSING

When an M*k*NN query is issued, we compute an initial $\lfloor \rho k \rfloor$ NN set, denoted by R, and the INS of R, $I(R)$. Here, $\rho \geq 1$ is a system parameter to balance the query result communication and recomputation costs. We call it the *prefetch ratio* and will evaluate its impact on the system performance in the experimental study. To improve the efficiency of computing $I(R)$, we precompute the Voronoi diagram of O and index it with an VoR-tree [20]. Since we do not need to compute the order-k Voronoi cells, our algorithm can answer kNN queries where k is determined at query time.

We return the set R and $I(R)$, where the top k objects of R are marked as the kNN set ($NN_k(q)$) and $I(R) \cup R \backslash NN_k(q)$ are marked as the IS. Then query maintenance starts. At each timestamp, we check whether the current kNN set is nearer to q than the IS: (i) if it is, then the current kNN set is still valid, as guaranteed by Theorem 2; (ii) if it is not, we perform the kNN set update procedure. If there are data object updates, we also update the kNN set and the IS according to the data object updates. Algorithm 1 summarizes the query maintenance process.

Algorithm 1: Query Maintenance

```
input : query object q, prefetched set R, current kNN set O' and its
        influential set IS
output: kNN set at each timestamp
1  while true do
2      r ← Validation(q, O', IS)
3      if r.isValid = false then
4          Update(q, R, O', IS, r.candidate, r.delete)
5      Process updates of O
```

6.1 *k*NN Set Validation

In Algorithm 1, we use the function `Validation` to test whether the current kNN set is nearer to q than the IS is. In the simplest way, this function can be implemented as a sequential scan on the kNN set and the IS, and find one data object in each of the two sets. In the kNN set we find the one that is the farthest from q, denoted by *r.delete*; in the IS we find the one that is the nearest to q, denoted by *r.candidate*. If *r.candidate* is closer to q than *r.delete*, the current kNN set has become invalid and we need to use the function `Update` to update the kNN set and IS.

6.2 *k*NN Set Update

When the kNN set validation fails, we need to update the kNN set and the corresponding IS: (i) If q has entered a neighboring order-k Voronoi cell, as Lemma 2 suggests, the new kNN set will differ from the current one by only one data object. 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 first check whether the new kNN set is still in R. If yes, then we just need to return this new kNN set. If not, we compute the new sets of R and $I(R)$ and return the new kNN set and IS.

Algorithm 2: Update

```
input : query object q, prefetched set R, current kNN set O' and its
        influential set IS, r.candidate, r.delete
output: new prefetched set R, kNN set O'' and its influential set IS'
1  O'' ← O' \ {r.delete} ∪ {r.candidate}
2  IS' ← (IS ∪ {r.delete} ∪ N_O(r.candidate)) \ O''
3  if O'' ≺_q IS' then return (R, O'', IS')
4  else
5      O'' ← kNN in R, IS' ← IS ∪ O' \ O''
6      if O'' ≺_q IS' then return (R, O'', IS')
7      else
8          Recompute R, I(R)
9          O'' ← top k in R, IS' ← R ∪ I(R) \ O''
10         return (R, O'', IS')
```

Since we do not compute the order-k Voronoi cells, we cannot determine whether q has entered a neighboring order-k Voronoi cell. However, we observe that, if the validation fails, we will obtain 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 $O'' = O' \cup \{r.candidate\} \setminus \{r.delete\}$. We test whether it is indeed the new kNN set by testing whether it is closer to q than its IS.

According to Lemma 5, the INS of O'' consists of the Voronoi neighbors of all objects in O''. We know that O' only differs from O'' by {r.candidate, r.delete}. Thus, we can derive an IS of O'' from the IS of O' as $IS(O'') = IS(O') \cup \{r.delete\} \cup N_O(r.candidate) \backslash O''$. Then we test whether $IS(O'') \circlearrowleft O''$ holds. (i) If yes then we know that O'' is the new kNN set, and we just need to return it as well as its IS. (ii) If not then we need to update the kNN set as described in the last paragraph. Algorithm 2 summarizes the procedure.

Note that we have precomputed the Voronoi diagram of O and stored an object with its Voronoi neighbors. When transferring the set R to the query object, we also transfer the corresponding Voronoi neighbors. Thus, no communication cost will incur unless IS needs to be updated with the Voronoi neighbors of *r.candidate* (line 2) if *r.candidate* $\notin R$, or the sets R and $I(R)$ need to be recomputed (line 8).

6.3 Data Object Update

When there are data object updates, we first update the Voronoi diagram and index structures on the data objects and then update the kNN set and the IS. We will focus on insertion and deletion, since a data object position update can be done by first deleting the data object and re-inserting the updated data object.

We use the standard R*-tree update algorithms [3] for updating the R*-tree index on the data objects and the VoR-tree update algorithms [20] for updating the Voronoi cells on the data objects. The kNN set and the IS are updated as follows.

6.3.1 *Insertion*

When a new data object p is added to the data set, there are two cases: (i) if it is nearer to the query object q than some current kNN, then we add it to the kNN set and update IS from the current kNN set, the current IS and p; (ii) otherwise, we simply add p to the IS

as this will not affect the correctness of the kNN set and the extra data objects in the IS will be filtered out next time the kNN set and IS are recomputed. Algorithm 3 summarizes the process.

Algorithm 3: Insertion

input : new data object p, prefetched set R, query object q, kNN set O' and its influential set IS
output: updated prefetched set R, kNN set O' and influential set IS

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'
4 **if** $d(p,q) < d(p_k,q)$ **then**
5 **return** $(R,\ O' \cup \{p\} \backslash \{p_k\},\ IS \cup \{p_k\})$
6 **return** $(R,\ O',\ IS \cup \{p\})$

Note that in the above insertion procedure, we do not need to add the Voronoi neighbors of p to IS when p is added to O'. Instead, we add p_k to IS. The correctness of doing so is guaranteed by the following theorem.

THEOREM 3. *Given the* k*NN set O' of the query object q with p_k as the k^{th} nearest neighbor, and a new data object p satisfying $d(p,q) < d(p_k,q)$, then for the set $O'' = O' \cup \{p\} \backslash \{p_k\}$,*

$$I(O') \cup \{p_k\} \circlearrowleft O''. \tag{13}$$

PROOF. According to Lemma 5, the INS of O'' consists of the Voronoi neighbors of all objects in O'', which suggests that we should add the Voronoi neighbors of p, $N_{O\cup\{p\}}(p)$, to the new IS. However, for each object $o \in N_{O\cup\{p\}}(p)$, we prove that if $o \notin I(O')$ then it does not contribute to the edges of $V^k(O'')$.

$$\begin{aligned} V^k(O'') &= \bigcap_{p_i \in O''} V_{(O\backslash O'')\cup\{p_i\}}(p_i) \\ &= \Big(\bigcap_{p_i \in (O'\backslash\{p_k\})} V_{(O\backslash O'')\cup\{p_i\}}(p_i)\Big) \cap V_{(O\backslash O'')\cup\{p\}}(p). \end{aligned}$$

1. In $\bigcap_{p_i\in(O'\backslash\{p_k\})} V_{(O\backslash O'')\cup\{p_i\}}(p_i)$, for each p_i, $o \notin I(O')$ implies that $o \notin N_{O\backslash O'\cup\{p_i\}}(p_i)$. By definition of the Voronoi cell we know that $b(o,p_i)$ will not be any edge of the Voronoi cell $V_{(O\backslash O'')\cup\{p_i\}}(p_i)$.
2. In $V_{(O\backslash O'')\cup\{p\}}(p)$, suppose $b(o,p)$ contributes as an edge of $V^k(O'')$. We can place a circle centered at a point o' on the edge with the radius being $d(o',o)$. Then, by Definition 3 we know that the data objects in $O'\backslash\{p_k\}$ are all enclosed by the circle while no data object in $O\backslash O'$ is. Thus, by Lemma 4, o must be a Voronoi neighbor of at least one data object in O', which contradicts to the assumption that $N_{O\cup\{p\}}(p) \not\subseteq I(O')$.

Therefore, the Voronoi neighbors of p which are not in $I(O')$ do not contribute to the edges of $V^k(O'')$ and hence can be safely discarded. □

6.3.2 Deletion

When a data object p is to be deleted, there are three cases: (i) p is in the current kNN set; (ii) p is in the current IS; (iii) p is not in either the current kNN set or the IS. Case (iii) does not involve kNN set or IS update and hence will not be discussed further. Processing of Cases (i) and (ii) is summarized in Algorithm 4.

(i) Deletion from the current kNN set: If $p \in O'$, then both O' and its IS need to be updated (lines 2 to 4). We find the nearest data object in the current IS to replace p in O' (line 3), the correctness of which is guaranteed by the following theorem.

THEOREM 4. *Given a* k*NN set O' and a set $IS \supseteq I(O')$, if a data object p in O' is deleted, then the new* k*NN set O' should be*

$$O' = O' \backslash \{p\} \cup \{p'\}, p' \in IS. \tag{14}$$

PROOF. Kolahdouzan and Shahabi [11] prove that the $(k+1)^{th}$ nearest neighbor of q must be a Voronoi neighbor of at least one data object in $NN_k(q)$. This means that the $(k+1)^{th}$ nearest neighbor of q is in $I(O')$ and hence in IS. When an object p in O' is removed, the $(k+1)^{th}$ nearest neighbor becomes the k^{th} nearest neighbor, which means that the new kNN set is formed by $O'\backslash\{p\} \cup \{p'\}$, where p' is an object in IS. □

After p' is added to O', according to Definition 8, its Voronoi neighbors are add to the IS to guarantee its completeness (line 4).

Algorithm 4: Deletion

input : data object to be deleted p, prefetched set R, query object q, current kNN set O' and its influential set IS
output: updated prefetched set R, kNN set O' and influential set IS

1 **if** $p \in R$ **then** $R \leftarrow R \backslash \{p\}$
2 **if** $p \in O'$ **then**
3 $p' \leftarrow$ nearest neighbor to q in IS
4 **return** $(R,\ O'\backslash\{p\} \cup \{p'\},\ IS \cup N_O(p') \backslash O' \backslash \{p'\})$
5 **else if** $p \in IS$ **then**
6 **return** $(R,\ O',\ IS \cup N_O(p) \backslash O' \backslash \{p\})$

(ii) Deletion from the current IS: If p belongs to the IS, then we can update the IS by first removing p and then adding all Voronoi neighbors of p which are not currently in O' or the IS (lines 6), as supported by the following theorem.

THEOREM 5. *Given a* k*NN set O' and a set $IS \supseteq I(O')$, if a data object p in IS is deleted, we have*

$$IS \cup N_O(p) \backslash O' \backslash \{p\} \circlearrowleft O'. \tag{15}$$

PROOF. Intuitively, when a data object p in IS is to be removed, we need its Voronoi neighbors to replace p to be the safe guarding objects. We omit the full proof due to space limit. □

7. COST ANALYSIS

In this section we present a comparative cost analysis on our method, denoted by INS-kNN, and the V*-Diagram algorithm [16]. Following V*-Diagram [16], we assume uniform data distribution in the analysis. We also assume that the query processor holds the set of data objects and reports the kNN sets to the query issuer.

Communication cost: We first analyze the communication frequency between the query processor and the query issuer, denoted by f_{INS} and f_{VD} for INS-kNN and V*-Diagram, respectively.

INS-kNN only needs to re-send the kNN set to the query issuer when $NN_k(q) \not\subseteq R$[2]. This happens only when the query object q moves out of the region defined by R, which consists of the order-k Voronoi cells of the data objects in R. According to [16], the communication frequency is inversely proportional to the root of the area of the safe region, which is the region defined by R in our case. Further, according to [18], there are $\mathcal{O}(k(n-k))$ order-k Voronoi cells on a set of n data objects. Thus, there are $\mathcal{O}(k(\lfloor \rho k \rfloor - k))$ order-k Voronoi cells on the data objects in R (recall that $\rho \geq 1$ is the prefetch ratio, and R contains $\lfloor \rho k \rfloor$ data objects). Let the size of the data space be 1. Then the area of the region that consists

[2]There are two cases, (i) the neighbors of *r.candidate* are sent (line 2 in Algorithm 2 when *r.candidate* $\not\subset R$), and (ii) updated sets of R and IS are sent (line 8 in Algorithm 2).

of all the order-k Voronoi cells of data objects in R is $\mathcal{O}(k(\lfloor\rho k\rfloor - k)/(k(n-k)))$, where n denotes the total number of data objects. Therefore, we have

$$f_{\text{INS}} = \mathcal{O}(\sqrt{(n-k)/(\lfloor\rho k\rfloor - k)}). \quad (16)$$

On the other hand, the updating frequency of V*-Diagram, f_{VD}, is $\mathcal{O}(\sqrt{kn/x})$ according to [16]. Note that $\lfloor\rho k\rfloor - k$ has the same meaning in INS-kNN as x in V*-Diagram. Therefore,

$$f_{\text{INS}}/f_{\text{VD}} = \mathcal{O}(\sqrt{\frac{n-k}{nk}}) < \mathcal{O}(1/\sqrt{k}). \quad (17)$$

From this inequality we can see that our algorithm is expected to have a smaller communication frequency than that of V*-Diagram. This is because the set R used in our algorithm conceptually defines a strict safe region while V*-Diagram uses an approximate safe region. However, in our experiments we observe that the actual communication cost of our algorithm may not always be smaller, especially when k is small and we do little prefetching, where the advantage of our algorithm becomes less significant. This is because that to maintain the strict safe region we need to send a larger number data objects to the query issuer each time the kNN set is recomputed and sent.

***k*NN set recomputation cost:** Every time the kNN set is to be recomputed, the query processor needs to compute in two steps, (i) the new set of R, which consists of the top $\lfloor\rho k\rfloor$ NNs, and (ii) its corresponding influential set IS.

(i) For the first step, using the Best-first [7] kNN algorithm, we can obtain the new set of R in $\mathcal{O}(\log n + \lfloor\rho k\rfloor)$ time on average.

(ii) For the second step, we have precomputed the Voronoi diagram on O, and stored the Voronoi cells in a VoR-tree [20]. Using this tree, the IS (INS, actually) of a data set can be obtained in time linear to its size [20].

From Equation 11 and Lemma 5, it can be seen that the size of an INS relies on the number of Voronoi neighbors of each data object p' in O'. Okabe et al. [18] proved that the average number of edges per Voronoi cell does not exceed 6, which means there are 6 Voronoi neighbors for each data object on average. Since R only have $\lfloor\rho k\rfloor$ data objects, the total number of Voronoi neighbors, and hence the size of IS, is within $6\lfloor\rho k\rfloor$ on average. Note that data objects nearby tend to share the same Voronoi neighbors, the actual size of the IS is expected to be even smaller than $6\lfloor\rho k\rfloor$. Therefore, we can update R in $\mathcal{O}(\log n + \lfloor\rho k\rfloor)$ time on average and its corresponding IS in $\mathcal{O}(\lfloor\rho k\rfloor)$ time.

Since the recomputation happens every time there is a kNN set update request from the query client, the recomputation frequency is the same as the communication frequency. Therefore, the overall kNN set recomputation cost of INS-kNN, denoted by c_{INS}, is

$$c_{\text{INS}} = \mathcal{O}\left(f_{\text{INS}}(\log n + \lfloor\rho k\rfloor)\right). \quad (18)$$

On the other hand, since V*-Diagram retrieves $(k+x)$ data objects with a BF-kNN call for kNN set recomputation, its recomputation cost $c_{\text{VD}} = \mathcal{O}(f_{\text{VD}}(\log n + k + x))$.

***k*NN set validation cost:** At each timestamp we validate the kNN set. As discussed in Section 6.1, the kNN set validation process is simply done by comparing the farthest object in the current kNN set to the query object, and the nearest object in the IS. By a sequential scan, INS-kNN needs $\mathcal{O}(\lfloor\rho k\rfloor)$ time for each validation. There is possible optimization here to devise new structures for maintaining the IS and achieve higher validation efficiency, which is left as future work.

V*-Diagram validates the safe region w.r.t. the k^{th} NN and scans a list of bisectors sequentially to validate the fixed rank region, which requires $\mathcal{O}(k+x)$ time [16].

Effect of ρ: From the discussion above, we have that the communication frequency, kNN recomputation cost and kNN validation cost are proportional to $\mathcal{O}(1/\sqrt{\rho-1})$, $\mathcal{O}(\rho/\sqrt{\rho-1})$ and $\mathcal{O}(\rho)$, respectively. Thus, we know that a larger ρ may results in smaller communication cost but larger kNN recomputation and validation costs. In the following section, we will quantify the effect of ρ by experiments.

8. EXPERIMENTS

In this section, we compare our algorithm with the state-of-the-art algorithm V*-Diagram [16] empirically. We present the experimental settings in Section 8.1, and the results in Section 8.2.

8.1 Settings

All experiments[3] are implemented using Scala programming language and conducted on a computer running OS X 10.9.3 with a 2.4 GHz Intel i5 CPU and 8GB memory.

Data sets. Both real and synthetic data sets are used in the experiments. We use two real data sets[4] called the Canadian Postal Codes data set and the Crowd Sourced Street Address data set, which contain 935,124 and 12,232,909 data objects, respectively. We also use two real trajectory sets[5] for the query object, called the Trucks data set and the Buses data set, respectively. They consist of 276 and 145 trajectories, respectively. Each trajectory consist of location information for a truck/bus within a day, collected every 30 seconds.

The synthetic data sets are generated with uniform distribution. We also generate two types of query objects: *random (denoted by "R")* and *directional (denoted by "D")*. A random query object updates its moving direction randomly at each timestamp; a directional query object randomly chooses a moving direction, and keeps moving along the direction until it hits the data space boundary, where it randomly chooses a new moving direction that will keep it inside the data space. We vary the speed of the query object from 1 to 256 units per timestamp, where the default value is 32.

We map the data sets on to a space of 20,000 $\times$ 20,000 square units. The VoR-tree [20] is used to index the data objects in each data set, where each data entry contains a data object and its Voronoi neighbors.

Parameters. To evaluate the algorithms under various settings, we vary the value of k from 5 to 50, the value of ρ from 1.5 to 5 and the size of the data set from 0.25 million to 1.5 million. By default, we set k at 10, ρ at 1 and use the two real data sets. The parameters are summarized in Table 2.

Table 2: System parameters

Parameter	Default	Range
Speed of the query issuer	32	1, 2, 4,$\cdots$, 256
k, number of query objects	10	5, 10, 15,$\cdots$, 50
x, number of auxiliary objects	9	3, 6, 9,$\cdots$, 18
ρ, prefetch ratio	1	1.5, 2, 2.5,$\cdots$,5
Data set size (million)	-	0.25, 0.5, 0.75, 1, 1.5

Baseline algorithm. We compare our method with the state-of-the-art algorithm V*-Diagram [16], as described in Section 3.2.

In the experiments, we use the client-server model, where the server holds the data set and is responsible for generating the kNN

[3] `https://github.com/chiewen/CkNN.git`

[4] `http://geocoder.ca`

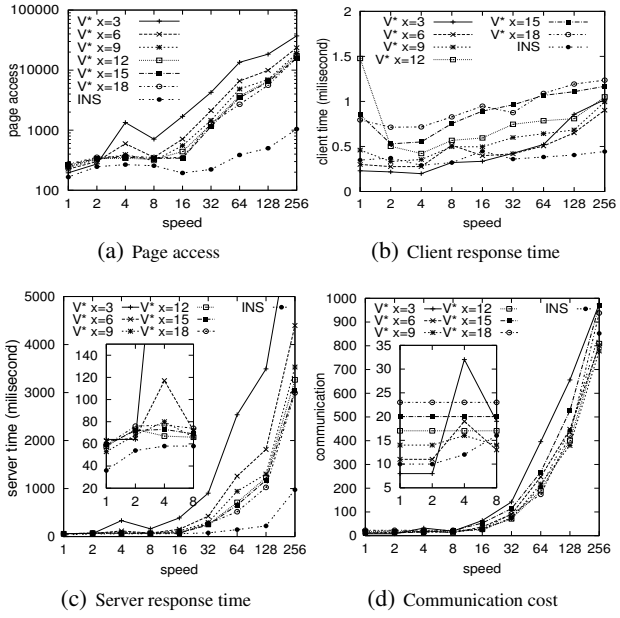
[5] `http://www.chorochronos.org/`

sets, while the clients are the query objects and they have the current kNN sets and IS. When the client updates its location, it first validates its kNN set, and only reports the update if the kNN set has become invalid. Changing the implementation to a centralized model can be done straightforwardly by moving the kNN set and IS to the server. We implemented both disk-based and memory-based versions of the proposed algorithm. In the experiments, we run the system for 300 timestamps and record the response time (on both the clients and the server, respectively), the number of page accesses (if disk-based implementation) and the communication cost between the query issuer and the query processor. Here, the communication cost is counted as the number of objects transferred from the query processor to the query issuer. We run each experiment 20 times and report the average recorded value.

8.2 Results

In this subsection we report the experimental results. In the figures we denote the proposed algorithm by "INS" and the baseline algorithm by "V*", 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. Figure 6 shows the algorithm performance where x is varied from 3 to 18 on the Postal Codes data set with a Random query object.



(a) Page access (b) Client response time

(c) Server response time (d) Communication cost

Figure 6: Optimal value of x

From the figure we can see that for all values of x tested, the proposed method outperforms V*-Diagram in terms of the page access number and the server response time. This is because the conceptual safe region defined by the IS of the proposed method is guaranteed to be as large as possible while the safe region of V*-Diagram is not. Meanwhile, we can compute the new IS from the current IS, which saves the server response time.

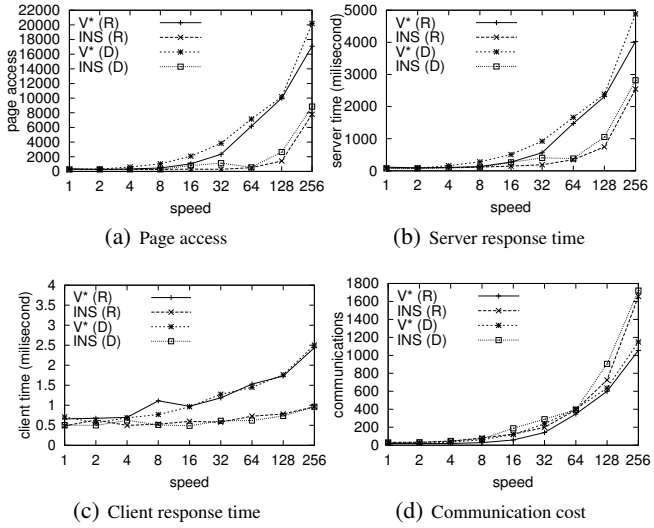
On the client side, the proposed method outperforms V*-Diagram in all cases, except for $x = 3, 6$ and the query object speed is below 8. This is because when x and the query object speed are both small, the safe region verification of V*-Diagram is relatively simple. However, as can be seen in Figure 6 (c), the server response time of V*-Diagram in these cases are higher, and the total response time of V*-Diagram is higher.

From Figure 6 we see that when $x = 9$ V*-Diagram shows the best overall performance. Experiments on other data sets show a similar pattern. We omit the figures due to space limit. In the following experiments, we will use $x = 9$ as the default value.

8.2.1 Varying Query Object Speed

We now compare the INS algorithm with the optimized V*-Diagram algorithm. We first vary the query object speed from 1 to 256 units per timestamp.

Disk-based implementation. Figure 7 shows the comparative performance of INS and V*-Diagram when the data reside on a hard disk for the Street Address data set. From Figure 7 (a) we see that as the query object speed increases, the number of page accesses increases for both algorithms. This is expected as higher query object speed means that the query object moves out of the safe regions more frequently. INS outperforms V*-Diagram constantly, and the advantage is up to 5 times. As predicted in the cost analysis, this due to a stricter safe region and simpler kNN set validation/update procedures.



(a) Page access (b) Server response time

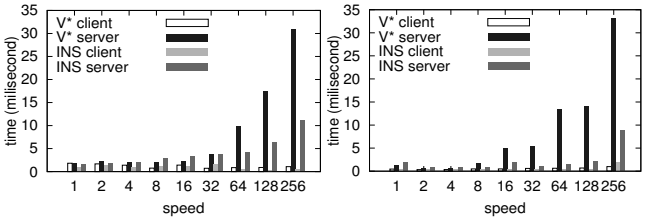
(c) Client response time (d) Communication cost

Figure 7: Varying query object speed

Since kNN computation causes most of the page accesses, which in turn causes most of the query processing time, Figures 7 (b) and (c) have similar patterns to that of Figure 7 (a) and INS outperforms V*-Diagram constantly in these figures. Note that Figure 7 (d) shows that the communication cost of INS is slightly higher than that of V*-Diagram. This is because, as discussed in the cost analysis, when both the prefetch ratio ρ and k are small ($\rho = 1$, $k = 10$), INS has a similar communication frequency to that of V*-Diagram. Meanwhile, INS has to transfer more data objects in each communication. Therefore, its communication cost is higher. We omit figures on other data sets as they show similar patterns. We do the same for the following subsections unless stated otherwise.

Memory-based implementation. Figure 8 shows the algorithm response time when the data reside in the memory for the Postal Codes data set. Again, INS outperforms V*-diagram in both server and client response time constantly. This is expected as INS shows lower CPU cost in the cost analysis due to simpler kNN set validation/update procedures.

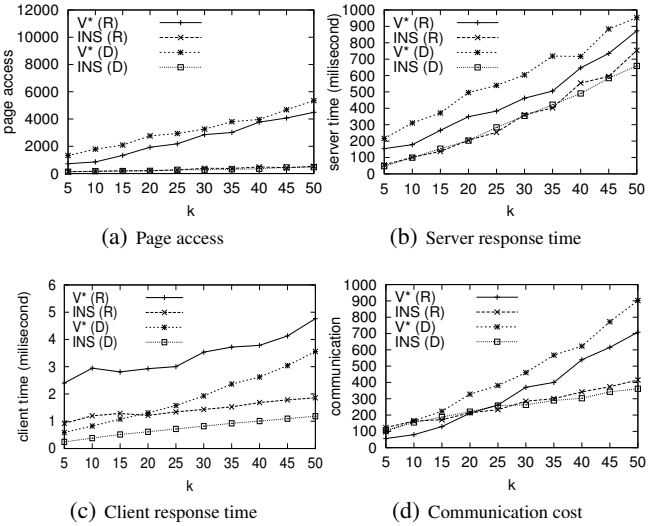
8.2.2 Varying k



(a) Postal Codes (Random) (b) Postal Codes (Directional)

Figure 8: Response time vs. query object speed (memory)

Next we evaluate the algorithms by varying k from 5 to 50. Figure 9 and 10 shows the results. We observe that while the costs of V*-Diagram increases quickly with the value of k, those of INS stay relatively stable. This is because INS computes the largest safe region possible and can utilize the current kNN set to derive the new kNN set, and hence it can constrain the number of kNN computation to the minimum. The results also show that, even though the communication cost of the proposed method is slightly higher when k is small, it increases more slowly than that of V*-Diagram. In Figure 10 we show the algorithm performance when the real trajectories are used. We can see that the they have similar patterns as those in Figure 9. These demonstrate the scalability of INS with respect to the change of the value of k as well as the trajectory of the query object.



(a) Page access (b) Server response time

(c) Client response time (d) Communication cost

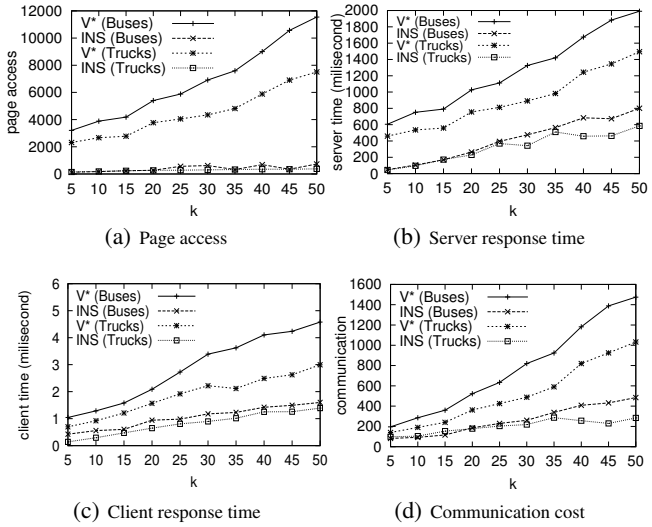
Figure 9: Varing k with generated trajectories

8.2.3 *Varying Data Set Cardinality*

In this set of experiments, we use synthetic data sets of different sizes, ranging from 0.25 million to 1.25 million, to evaluate the scalability of INS with respect to data set cardinality. The results in Figure 11 show that the performance of INS is much more stable compared with that of V*-Diagram with various values of x. This again confirms the superiority of the proposed algorithm.

8.2.4 *Effect of ρ*

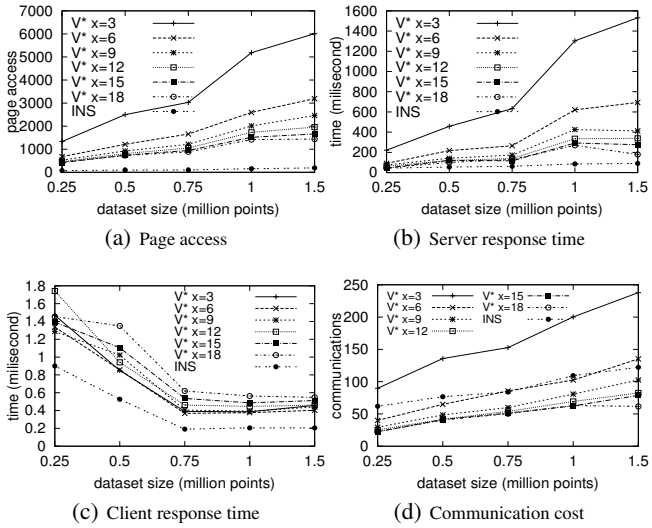
In our proposed method we have a parameter $\rho \geq 1$ to balance the computation and communication costs. In kNN set recomputation, we compute $\lfloor \rho k \rfloor$ NNs and use the extra $(\lfloor \rho k \rfloor - k)$ objects as a kNN "cache". We evaluate the effect of ρ by varying its value



(a) Page access (b) Server response time

(c) Client response time (d) Communication cost

Figure 10: Varing k with real trajectories



(a) Page access (b) Server response time

(c) Client response time (d) Communication cost

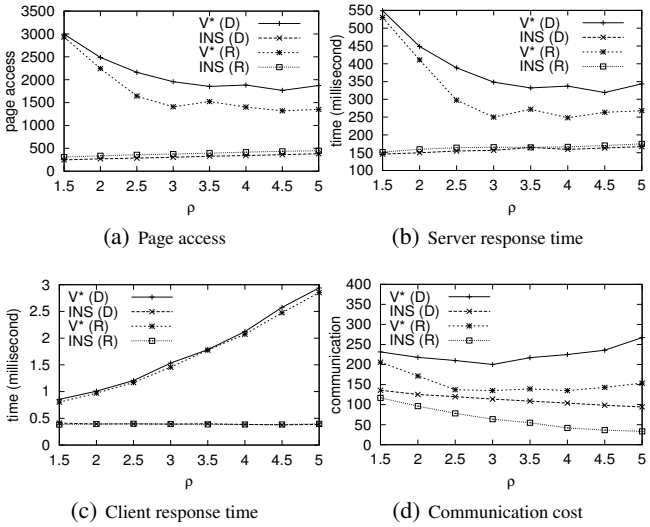
Figure 11: Varying data set cardinality

from 1.5 to 5. Correspondingly, we set the auxiliary object number x of V*-Diagram to be $\lfloor \rho k \rfloor - k$ so that INS and V*-Diagram have that same size of kNN "cache".

Figure 12 shows the result on the Street Address data set. It shows that INS outperforms V*-Diagram under various values of ρ. As expected, when ρ increases, the communication cost of INS decreases. Meanwhile, the other costs of INS increase but with a much slower speed.

9. CONCLUSIONS

In this paper, we revisited the moving k nearest neighbor (MkNN) query and presented an algorithm that processes the MkNN query efficiently. This algorithm takes 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 (no more than $6k$ on average) of data objects and can be used to validate the current kNN set. We replace the computation of order-k Voronoi cells with the computation of influential neighbor sets,



(a) Page access (b) Server response time

(c) Client response time (d) Communication cost

Figure 12: Effect of ρ

which is much more efficient. Our influential neighbor set based algorithm also supports 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 our algorithm outperforms the state-of-the-art algorithm on both I/O and computational costs consistently.

Acknowledgement. This work is supported by the National Basic Research Program of China under Grant No. 2012CB316201, the National Natural Science Foundation of China under Grant No. 61300021 and 61033007, Australian Research Council (ARC) Discovery Project DP130104587, Australian Research Council (ARC) Future Fellowships Project FT120100832, the Fundamental Research Funds for the Central Universities of China No. N120304003, National Key Technology R&D Program of China 2012BAK24B01 and China Scholarship Council.

10. REFERENCES

[1] M. E. Ali, E. Tanin, R. Zhang, and L. Kulik. A motion-aware approach for efficient evaluation of continuous queries on 3D object databases. *VLDBJ*, 19(5):603–632, 2010.

[2] A. M. Aly, W. G. Aref, and M. Ouzzani. Spatial queries with two knn predicates. *PVLDB*, 5(11):1100–1111, 2012.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[4] R. Benetis, C. S. Jensen, G. Karĉiauskas, and S. Ŝaltenis. Nearest and reverse nearest neighbor queries for moving objects. *VLDBJ*, 15(3):229–249, 2006.

[5] Y. Gao and B. Zheng. Continuous obstructed nearest neighbor queries in spatial databases. In *SIGMOD*, pages 577–590, 2009.

[6] Y. Gao, B. Zheng, W.-C. Lee, and G. Chen. Continuous visible nearest neighbor queries. In *EDBT*, pages 144–155, 2009.

[7] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *SSD*, pages 83–95, 1995.

[8] H. Hu, J. Xu, and D. L. Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *SIGMOD*, pages 479–490, 2005.

[9] X. Huang, C. S. Jensen, and S. Šaltenis. The islands approach to nearest neighbor querying in spatial networks. In *SSTD*, pages 73–90. 2005.

[10] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang. Effective density queries on continuously moving objects. In *ICDE*, pages 71–82, 2006.

[11] M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.

[12] L. Kulik and E. Tanin. Incremental rank updates for moving query points. In *GIS*, pages 251–268. 2006.

[13] C. Li, Y. Gu, F. Li, and M. Chen. Moving k-nearest neighbor query over obstructed regions. In *Asia-Pacific Web Conference (APWEB)*, pages 29–35. IEEE, 2010.

[14] C. Li, Y. Gu, G. Yu, and F. Li. wneighbors: a method for finding k nearest neighbors in weighted regions. In *DASFAA*, pages 134–148. Springer, 2011.

[15] C.-H. Liu, E. Papadopoulou, and D.-T. Lee. An output sensitive approach for the L1/L∞ k-nearest-neighbor voronoi diagram. In *Algorithms–ESA 2011*, pages 70–81. 2011.

[16] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. The v*-diagram: a query-dependent approach to moving knn queries. *PVLDB*, 1(1):1095–1106, 2008.

[17] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. Analysis and evaluation of v*-knn: an efficient algorithm for moving knn queries. *VLDBJ*, 19(3):307–332, 2010.

[18] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams*, volume 501. Wiley. com, 2009.

[19] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD Rec.*, volume 24, pages 71–79, 1995.

[20] M. Sharifzadeh and C. Shahabi. Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. *PVLDB*, 3(1-2):1231–1242, 2010.

[21] Z. Song and N. Roussopoulos. K-nearest neighbor search for moving query point. In *SSTD*, pages 79–96, 2001.

[22] Y. Tao and D. Papadias. Time-parameterized queries in spatio-temporal databases. In *SIGMOD*, pages 334–345, 2002.

[23] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298, 2002.

[24] A. Y. Xue, R. Zhang, Y. Zheng, X. Xie, J. Huang, and Z. Xu. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *ICDE*, pages 254–265, 2013.

[25] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based spatial queries. In *SIGMOD*, pages 443–454, 2003.

[26] R. Zhang, H. V. Jagadish, B. T. Dai, and K. Ramamohanarao. Optimized algorithms for predictive range and knn queries on moving objects. *Information Systems*, 35(8):911–932, 2010.

[27] R. Zhang, D. Lin, R. Kotagiri, and E. Bertino. Continuous intersection joins over moving objects. In *ICDE*, pages 863–872, 2008.

[28] R. Zhang, J. Qi, D. Lin, W. Wang, and R. C.-W. Wong. A highly optimized algorithm for continuous intersection join queries over moving objects. *VLDBJ*, 21(4):561–586, 2012.