INSQ: An Influential Neighbor Set Based Moving *k*NN Query Processing System

Chuanwen Li*, Yu Gu*, Jianzhong Qi†, Ge Yu*, Rui Zhang† and Qingxu Deng*
*College of Information Science and Engineering
Northeastern University
Email: {lichuanwen,guyu,yuge,dengqx}@ise.neu.edu.cn
†Department of Computing and Information Systems
University of Melbourne
Victoria, Australia
Email: {jianzhong.qi,rui.zhang}@unimelb.edu.au

***Abstract*—We revisit the moving *k* nearest neighbor (M*k*NN) query, which computes one's *k* nearest neighbor set and maintains it while at move. Existing M*k*NN algorithms are mostly safe region based, which lack efficiency due to either computing small safe regions with a high recomputation frequency or computing larger safe regions but with a high cost for each computation. In this demonstration, we showcase a system named INSQ that adopts a novel algorithm called the Influential Neighbor Set (INS) algorithm to process the M*k*NN query in both two-dimensional Euclidean space and road networks. This algorithm uses a small set of safe guarding objects instead of safe regions. As long as the the current *k* nearest neighbors are closer to the query object than the safe guarding objects are, the current *k* nearest neighbors stay valid and no recomputation is required. Meanwhile, the region defined by the safe guarding objects is the largest possible safe region. This means that the recomputation frequency is also minimized and hence, the INS algorithm achieves high overall query processing efficiency.**

I. Introduction

Location-based services (LBS) are more and more popular as smart mobile devices have become prevalent. On-going efforts have been made to improve the user experience of mobile LBS through improvements in moving query processing efficiency. In this paper we revisit a major type of moving query, the *moving k nearest neighbor (M*k*NN)* query [1], [4]. *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, a M*k*NN query can be used to report the 3 nearest gas stations continuously while one drives on a highway, or the 5 nearest points of interest (POI) continuously while a tourist is walking around a city.

Studies on the M*k*NN query have mainly used the *safe region based* approach [5], [6]. The idea is based on that objects at nearby positions often share the same *k*NN set. There may be a region where all points have the same *k*NN set. Inside such a region, an object is "safe" to move freely without causing its *k*NN set to change. Thus, such a region is called a safe region. Using the safe region significantly reduces the M*k*NN query processing cost because the *k*NN 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 *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 object location updates.

Existing methods either fall short in construction overhead or validation overhead. Specifically, earlier studies [2], [6] use *Voronoi cells* [6] as the safe regions, which have high construction overhead. A more recent study [5] uses relaxed safe regions to reduce the construction overhead, but it has to recompute the safe regions more frequently and has higher validation overhead.

A Voronoi cell based safe region is essentially an *order-k Voronoi cell*. Given a *k*NN set, its order-*k* Voronoi cell is defined as a region where the *k*NN set stays valid [6]. The computation cost of computing order-*k* Voronoi cells on the fly is prohibitively high since it requires a combination of *k* order-1 Voronoi cells. Precomputing the order-*k* Voronoi cells [2] is also unpractical due to the rapid increase in the number of order-*k* Voronoi cells as *k* increases.

In this demonstration, we showcase an Influential Neighbor Set based system named *INSQ*. The influential neighbor set (INS) is an algorithm to avoid the high cost of M*k*NN query processing based on our previous paper [3], which uses *safe guarding objects* instead of safe regions. The intuition is that, since the query object moves continuously, when the *k*NN set changes, some data object near the current *k*NN objects must become one of 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, the current *k*NN set stays valid and 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 *k*NN set recomputation and hence minimum communication between the query client and the query processor, which is critical in LBS. To improve efficiency we identify a special set of safe guarding objects and call it the *influential neighbor set (INS)*, which can be computed and validated in time linear to *k*. This set reduces both construction and validation overhead at the same time.

We build an interactive demonstration program that shows the internal mechanism of the INS algorithm. It simulates a moving query object and displays the changing kNNs 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. *Note that in addition to the two-dimensional Euclidean space scenario which was discussed in our previous paper [3], we also discuss and demonstrate how the INS algorithm works in road networks in this paper.*

II. Influential Neighbor 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 the set of safe guarding objects an *influential set (IS)*, in the sense that they are *influential* in determining whether a kNN set is valid. The *influential neighbor set (INS)* is a special IS that can be efficiently computed, which will be detailed below. Now we first formalize the IS.

Definition 1: (Influential Set, IS) Given a set of data objects O, a query object q and a kNN set $O' \subset O$, we call a set $S \subseteq O \backslash O'$ an *influential set* of O', denoted by $S \circlearrowleft O'$, if O' stays as the kNN set of q as long as the objects in O' are closer to q than the objects in S are, i.e.,

$$O' = NN_k(q) \iff O' \prec_q S. \tag{1}$$

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.

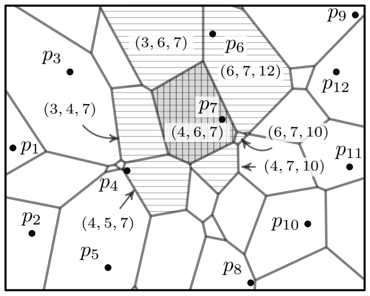


Fig. 1. The minimal influential set (MIS) of $O' = \{p_4, p_6, p_7\}$

Trivially, $O \backslash O'$ 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 is expensive. We aim to find the *minimal influential set*, which is the smallest influential set that can guarantee the validity of the current kNN set.

Definition 2: (Minimal Influential Set, MIS) Given a kNN set O', the *minimal influential set* (MIS) of O' is

$$MIS(O') = \Big(\bigcup_{V^k(O') \cap V^k(O'') \neq \emptyset} O'' \Big) \backslash O'. \tag{2}$$

Here, $V^k(O'') \cap V^k(O') \neq \emptyset$ means that the two order-k Voronoi cells are neighboring cells, i.e., they share an edge. Figure 1 gives an example, where $O' = \{p_4, p_6, p_7\}$ and the cross-lined region denotes $V^3_O(O')$. There are five neighboring order-3 Voronoi cells, as denoted by the horizontal-lined regions. The triples associated with 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', $MIS(O') = \{p_3, p_5, p_{10}, p_{12}\}$.

In [3] we prove that MIS is minimal. However, similar to computing the strict safe region (i.e., an order-k Voronoi cell), computing the MIS is an expensive operation. To reduce the computational cost, we compute an influential set that is slightly larger than the MIS but can be computed much more efficiently. By definition of the influential set, this will not affect the strictness in validating the kNN sets.

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

Definition 3: (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., $V(p_i) \cap V(p_j) \neq \emptyset$. 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 [7]. *The influential neighbor set is the union of the order-1 Voronoi neighbor sets of all current kNNs.*

Definition 4: (Influential neighbor set, INS) Given a kNN 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 $p \notin O', \exists p' \in O' : V(p) \cap V(p') \neq \emptyset$. We call the set of all influential neighbors of O' the *influential neighbor set (INS)* of O', denoted by $I(O')$,

$$I(O') = \Big(\bigcup_{p' \in O'} N_O(p') \Big) \backslash O'. \tag{3}$$

In [3] we prove that an INS is an IS, which guarantees the correctness of using the INS for query processing.

III. Query Processing

When a MkNN 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 have discussed how to choose its value in [3]. To improve the efficiency of computing $I(R)$, we precompute the Voronoi diagram of O and index it with an VoR-tree [7].

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; (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.

A. kNN Set Validation

To validate the current kNN set, we scan the kNN set and the IS. 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 update it.

B. kNN Set Update

There are two cases for kNN set update: (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. 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.

IV. INS in Road Networks

We extend the influential neighbor set to road networks in this demonstration paper. We consider a planar undirected (connected) graph $G = \langle V, E \rangle$ formed by a set of vertices $V = \{v_1, v_2, \ldots, v_m\}$ and a set of edges $E = \{e_1, \ldots, e_l\}$. We assume that the data objects $O = \{p_1, \ldots, p_n\}$ are all at the vertices (otherwise we can add them to the set of vertices and add edges accordingly).

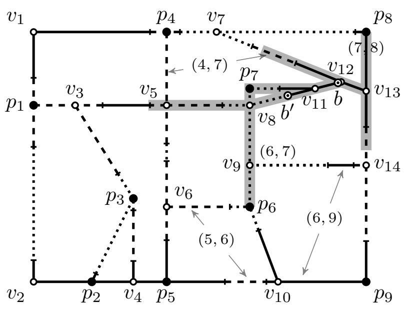


Fig. 2. A road network example

The concept of Voronoi diagram still applies in road networks. As such, we can show that the influential neighbor set defined by Definition 4 is still a super set of the MIS defined by Definition 2 in road networks.

Theorem 1: In a road network, given a kNN set O_{knn}, $I(O_{knn})$ as defined by Definition 4, and $MIS(O_{knn})$ as defined by Definition 2:

$$MIS(O_{knn}) \subseteq I(O_{knn}). \quad (4)$$

The detailed proof is omitted due to space limit. We use Fig. 2 to illustrate the idea. The figure shows an order-2 Voronoi diagram in a road network. 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 that belong to the same order-2 Voronoi cell by the same line type. There is a number pair, e.g., $(6,7)$, besides these edge segments, meaning that these edge segments belong to an order-2 Voronoi cell, e.g., $V^2(p_6, p_7)$.

Let the current kNN set be $O_{knn} = \{p_6, p_7\}$. The MIS by definition should be $MIS(O_{knn}) = \{p_4, p_5, p_8, p_9\}$. For each pair of objects $(p, p') \in O_{knn} \times MIS(O_{knn})$, e.g., (p_7, p_8), we can find a "*mid-point*" on the shortest path between p_7 and p_8. For example, the mid-point between p_7 and p_8 is denoted by b in the figure. This mid-point satisfies $d(b, p_7) = d(b, p_8)$; no other object in $O \backslash O_{knn} \backslash I(O_{knn})$ is nearer to b than p_7 or p_8. Here, $d()$ denotes the shortest road network distance between two objects. The existence of this mid-point indicates that p_7 and p_8 are also order-1 Voronoi neighbors, which means that p_8 should also belong to $I(O_{knn})$. Thus, we have shown that every object in $MIS(O_{knn})$ also belongs to $I(O_{knn})$, i.e., $MIS(O_{knn}) \subseteq I(O_{knn})$.

The above theorem allows to validate the kNN set using the INS in road networks. However, in road networks, checking whether the query object is nearer to any object in the INS than the objects in the kNN set is not a trivial operation. This is because it requires network distance (shortest path) computation. To constraint the search space of this computation, we have proven the following theorem.

Theorem 2: Given a Voronoi diagram D_O on a road network with a set of data objects O, a 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}.

This theorem guarantees that we just need to consider the (smaller) road network formed by the current kNN set and the INS for query validation. We omit the detailed proof due to space limit. Based on the two theorems above, we have developed the INS algorithm for road networks which is showcased in the following section.

V. Demonstration

The system[1] is implemented as a Scala Swing application. The application runs in two modes, *Road Network* mode and *2D Plane* (Euclidean space) mode. Its user interface consists of two panels (cf. Figure 3):

(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 *Road Network* setting and the *2D Plane* setting.

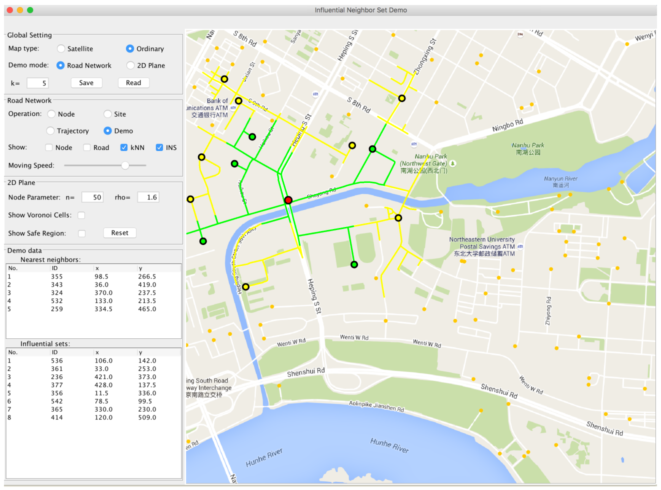
The global setting includes the underlying map to be used, the data space to be used, and the value of the query parameter k. There are also two buttons "Save" and "Read" which are for recording and loading the demonstration settings.

The road 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 MkNN query will be simulated. The setting also includes options to control which objects 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 kNN set and the INS are displayed.

[1]Source code available at https://github.com/chiewen/CkNN.git



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. 3. Screenshots (Road Network, $k = 5$)

(ii) *Main panel.* This is the right panel of the user interface. It displays a map of a road 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 Road Network mode, this trajectory must confine the underlying road 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 kNN set and the INS set, which are shown as the green dots and the yellow dots, respectively.

In the Road Network mode, the Voronoi cells of the objects in the kNN set and the INS are represented by the sets of green and yellow network edges, respectively.

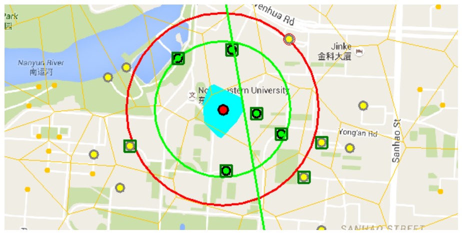
In the 2D Plane mode (Fig. 4), the data objects are surrounded by their respective order-1 Voronoi cells. The ones enclosed by green squares represent the objects in R. We use the cyan polygon to represent the current order-k Voronoi cell, which will turn red when it becomes invalid. We circle two special objects, the farthest object to q in the kNN 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 kNN set, while the red circle should never enclose any object in the INS, as long as the current kNN set is still valid.

Demonstrated Scenarios: We take the 2D Plane mode as an example. Fig. 4 displays two screenshots of the demonstration program, where each shows: (i) the query object stays in the order-k Voronoi cell of the current kNN set (i.e., the kNN set is valid), and (ii) the query object has moved out of the order-k Voronoi cell (i.e., the kNN set is invalid).

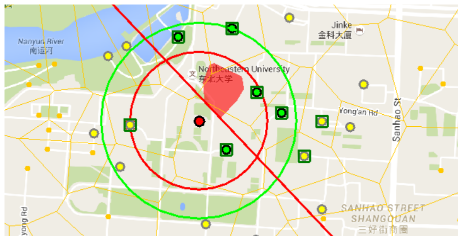
When the query object moves out of the order-k Voronoi cell shown in Fig. 4 (a) and moves into the position shown in Fig. 4 (b), the current kNN set becomes invalid since the green circle is now enclosed by the red circle (i.e., the farthest object to q in the kNN set is farther to q than the nearest object to q in the INS).

VI. Conclusion

We built and showcased a system named INSQ to process the MkNN query on both Euclidean space and road networks, based on a highly efficient algorithm INS. The program simulates the movement of the query object along the trajectory and displays the changing kNNs and the corresponding influential neighbors which are the key feature of the INS algorithm. Users can customize the behavior of the simulation and observe the mechanisms of the algorithm in detail. The source code of the INS algorithm and the INSQ system is publicly accessible through the Internet, which can be easily adapted to real systems such as smart phone applications.



(a) The kNN set is valid



(b) The kNN set is invalid

Fig. 4. Screenshots (2D plane, $k = 5$ and $\rho = 1.6$)

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant No. 61300021, 61472071, 61472072, 61402155, the National Basic Research Program of China (973 Program) under Grant No. 2012CB316201 and 2014CB360509, the Fundamental Research Funds for the Central Universities of China No. N140404008. Jianzhong Qi is the recipient of a Melbourne School of Engineering Early Career Researcher Grant (project reference number 4180-E55), and a University of Melbourne Early Career Researcher Grant (project number 603049). Rui Zhang is supported by ARC Future Fellow project FT120100832.

References

[1] T. Hashem, L. Kulik, and R. Zhang. Countering overlapping rectangle privacy attack for moving knn queries. *Information Systems*, 38(3):430–453, 2013.

[2] L. Kulik and E. Tanin. Incremental rank updates for moving query points. In *Geographic Information Science*, pages 251–268. 2006.

[3] C. Li, Y. Gu, J. Qi, G. Yu, R. Zhang, and W. Yi. Processing moving knn queries using influential neighbor sets. *PVLDB*, 8(2):113–124, 2014.

[4] T. Nguyen, Z. He, R. Zhang, and P. Ward. Boosting moving object indexing through velocity partitioning. *PVLDB*, 5(9):860–871, 2012.

[5] 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.

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

[7] 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.