Reverse *k*NN Query Algorithm on Road Network Distance

Tin Nilar Win, HtooHtoo, Yutaka Ohsawa
Graduate School of Science and Engineering, Saitama University, Japan
tinnilar.win@gmail.com, htoohtoo@mail.saitama-u.ac.jp, ohsawa@mail.saitama-u.ac.jp

Abstract

This paper proposed a reverse k nearest neighbor (RkNN) query algorithm in road network distance by using the simple materialization path view (SMPV) data structure.When a set of interest objects P and a query point q are given, RkNN query retrieves reverse nearest neighbors of q from the set P. Two types of RNN, monochromatic (MRNN) and bichromatic (BRNN)are classified in the literature. In conventional approaches for RNN query on a road network distance, it takes very long processing time becausekNN search algorithm is invoked on every visited node. Using SMPV in combination with incremental Euclidean restriction (IER) framework reduces processing time in kNN search significantly. This paper studied for both types of RNN comparing with the conventional method, Eager algorithm. With extensive experiments, the proposed method outperformed Eager algorithm in term of processing time especially when the k value is large.

1. Introduction

An increase of mobile applications highly depends on variety of spatial queries for location based services (LBS). Subsequently, there has been a great deal of researches on reverse *k* nearest neighbor query algorithms.Most existing algorithm are based on Euclidean distance, however in LBS application, especially in mobileenvironment, road network based queries are practically required. Query algorithm for R*k*NN based on Euclidean distance has been actively studied.

When a set of interest objects *P* and a query point *q* is given, a query to find the nearest neighbor of a point *q* (∈ P) is called a nearest neighbor (NN) query. Reversely, when the NN of *p* (∈ P) is *q* (∈ P), *p* is called a reverse nearest neighbor (RNN) of *q*.RNN query is returned as a result set. Figure 1 describes the example of NN and RNN for a game application in which players find their NN to shoot and reverse nearest neighbors (RNN) to avoid shoots from them. In this figure, circles represent players, and R1NNand R2NN of each player are listed.

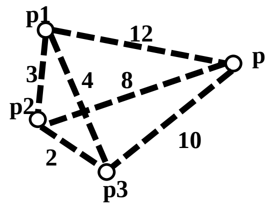

Figure 1. The example of NN vs RNN

Table 1. NN vs RNN

	1NN	2NN	R1NN	R2NN
p1	p2	p2,p3	ϕ	{p2, p3}
p2	p3	p3,p1	{p3,p1,p4}	{p3,p1,p4}
p3	p2	p2,p1	{p2}	{p2,p1,p4}
p4	p2	p2,p3	ϕ	ϕ

In general, R*k*NN query can be definedasthe following:

$$\text{R}k\text{NN} = \{p \in \text{P} | d(p,q) \leq d(p,p_k(p))\} \qquad (1)$$

where$p_k(p)$ is the k^{th} NN of *p*. When a set of interest points *P* and a query point *q* are given, an R*k*NN query retrieves the points which arenearest neighbors to *q*.

RNN query is classified as monochromatic and bichromatic RNN. In monochromatic (MRNN), a given set of interest points and a query point is the same data object.In contrast, for bichromatic (BRNN) query, two different data sets are given for the interest objects (*P*) and the query points (*S*) sets. Practically, this type of query is required in manyLBS related applications both for emergency and non-emergency cases, for instance, taxi allocation, facility management.Although very limited research has been focused on R*k*NN query on road network distance, existing approaches have shortcomings in processing times especially when the interest points are sparsely allocated on road network or when the value of *k* is large.

In this paper, we propose a fast R*k*NN query in road network distance using simple materialized path view (SMPV) data [2]. Our proposed algorithm for both MRNN and BRNN is compared with the existing approach, Eager algorithm proposed by Yiu et al. [1], and evaluated the performance of the proposed method. The proposed algorithm searches R*k*NN approximately two orders of magnitude faster than the existing work in terms of processing time. It especially outperforms when interest points are sparsely distributed in the road network and the value of *k* is large, and offers the stability in processing time and independency of the point distribution density.

The rest of the paper is organized as follows. Related work is described in Section 2. In Section 3, the SMPV data structure and shortest path search algorithm are discussed. We also describe the principles for an R*k*NN query on road network distance and proposed a fast query method in Section 4. Experimental results are presented in Section 5, and we conclude our paper in Section 6.

2. Related Work

In earlier literature, Euclidean distance based R*k*NN queries have been addressed. The concept of RNN is formally introduced by Korn et al. [3]. In their approach, the distance from each interest object of *P* to its NN is pre-computed. Given this data, a set of points and their distances to the NN are registered in an R-tree, and the circle centered at a data point with a radius equal to the distance to the NN is called its vicinity circle. The RNN of the query object *q* is found in the R-tree by searching the set of interest objects for those whose vicinity circles overlap with *q*. However, this method is not suitable for an R*k*NN query because *k* in an R*k*NN query is normally set when a query is issued. In their concept, the R-tree is constructed using vicinity circles of predefined k^{th} NN distances, and the distance to the k^{th} NN cannot be practically determined for R*k*NN before invoking the query.

Stanoi et al. [4] proposed RNN algorithm in which RNN search region is divided into six regions centered at the query point. Then, the set of data points *P* which is NNs of *q* are retrieved from each region.Tao et al. [5] proposed another efficient algorithm called TPL that recursively prunes the search space using the bisector between a query point *q* and its NN.These methods do not require any pre-computation. Therefore, they are applicable to general R*k*NNqueries, however, these efficient methods cannot be directly applied to R*k*NN queries in road network distances.

Yiu et al. [1] proposed the first R*k*NN algorithms applicable to road networks. The intuition behind is that the area is gradually enlarging by Dijkstra's to find for inclusion of R*k*NN in it. They proposed two algorithms (called the Eager and Lazy algorithms) that differ in their respective pruning methods. In these methods, the Eager algorithm searches R*k*NN significantly faster, especially when the value of *k* is small and the set of points are densely distributed. In contrast, for large *k* or sparsely distributed points, it requires long processing times because the Eager algorithm gradually enlarges the search area, similar to Dijkstra's algorithm, and the *k*NNs are searched at every visited road network node. To cope with this performance problem, Yiu et al. also proposed a path materialization method. For BR*k*NN query, Kornet al. [3] first proposed for Euclidean distance, and then Yiu et al. [1] researched Eager algorithm for BR*k*NN in road network distance. Even though BR*k*NN has been focused, respective approach was for BR1NN, and most traditional methods for BR*k*NNhas deficiency in processing time.

3. Simple Materialized Path View

3.1. Generating Distance Table

The principle behind the simple materialized path view (SMPV) is to partition a given graph G into several sub graphsbythe dotted lines as shown in Figure 2, called partitioned graphs (PGi) in this section onwards.

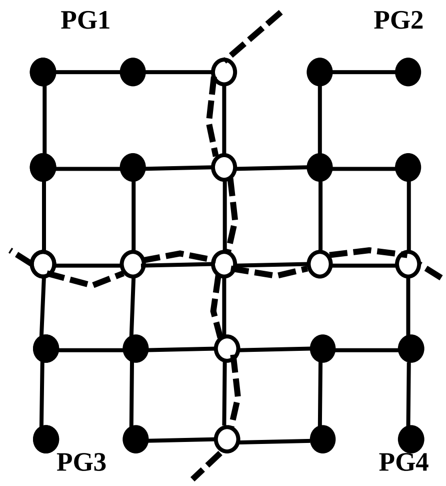

Figure 2.The flat graph and its partitions

The detail concept of SMPV is introduced in [2]. In this section, some modification of SMPV inwhich appended inner-to-border-nodes distance table is briefly presented.

Figure 2 is partitioned into four partitioned graphs. In this figure, white circles are called border nodes which belong to at least two PGs. Black circles are called inner nodes which are other nodes except border nodes. Two PGs are defined as adjacent PGs if they have at least one common border node. However, each edge belongs to only one PG exactly.

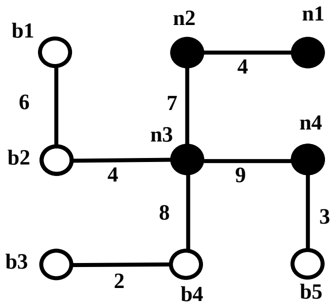

Figure 3.Extracted graph PG2

In the R*k*NNquery algorithm, a process to find the road network distance between two points (start and destination) is necessary. In this process, the pre-computed distance tables with SMPV data, border-to-border node Table 2(BBDT) and inner- to-border node Table 3.

Table 2. Border-to-border node table

	b1	**b2**	**b3**	**b4**	**b5**
n1	**21**	**15**	**21**	**19**	**23**
n2	**17**	**11**	**17**	**15**	**19**
n3	**10**	**4**	**10**	**8**	**12**
n4	**19**	**13**	**19**	**17**	**3**

Table 3. Inner-to-border-node table

	b1	**b2**	**b3**	**b4**	**b5**
b1	0	6	20	18	22
b2	6	0	14	12	16
b3	20	14	0	2	22
b4	18	12	2	0	20
b5	22	16	22	20	0

(IBDT) are applied.Table 2 shows the shortest path length between every pair of border nodes of PG_2shown in Figure 3.These lengthsare calculated by traveling inside the partitioned graph.If a path between a pair of nodes inside the partitioned graph is not connected, the infinity value is assigned.

The real road network is not always symmetrical because there might exist one-way roads or delays affecting only one direction of a two-way road.Thus, the transport matrix as shown in Table 3 is also prepared to retrieve the distance from an inner node as a starting point to a border node.

3.2. Partitioning a large graph

The real road network can be divided intopartitioned graphs by (1) selecting source nodes on the given road network, and (2) applying Dijkstra's multi-source shortest path algorithm, each road network is checked whether it is nearest from each source node, and then nearest nodes are grouped intosame partitioned graph.Then, BBDT and IBDT tables are prepared for each partitioned graph.

4. R*k*NNQuery

In this section, a basic method forR*k*NN query in road networks by applying an improved method based on the incremental Euclidean restriction (IER) framework is presented.

Lemma 1 Let*q*be a query point, *n* be a road network node and *p* be a data point that satisfies $d_N(q,n)>d_N(p,n)$. For any data point $p'(\neq p)$ whose shortest path to *q* passes through *n*,$d_N(q, p')>d_N(p,p')$. This means that *p* is not an RNN of *q*.

Yiu et al. [1] presented the lemma mentioned above and it is proved where $d_N(a,b)$ denotes the road network distance between *a* and *b*.

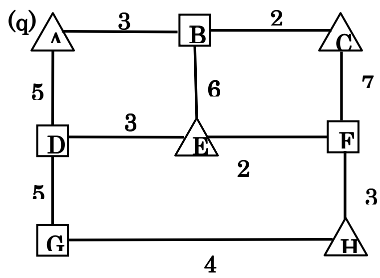

Figure 4. RNN query on a road network

Figure 4 shows an RNN query on simple road network. In this figure, rectangles represent road network nodes and triangles indicate data points. Data points are assumed to be located on nodes, but this restriction can be relaxed easily. The numbers assigned on edges are distances. To consider RNN on this figure, it is necessary to find nodes that are nearest neighbor (NN) to a query point *q* at point A. When we observe *D*, the NN data point of *D* is *E* and the NN data point of *E* is *H*. Therefore, *A* is not the NN data point of *E*. If we substitute *n* with *D*, *p* with *E*, *p′* with *H* in Lemma 1, we obtain the relations $d_N(A,D)>d_N(E,D)$ and $d_N(A,H)>d_N(E,H)$. Therefore, even if we continue searching beyond*D*, we cannot find the RNN of *q*.

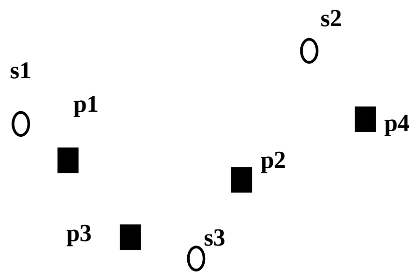

Figure 5.The example of BRNN query

Alternatively, Figure 5 shows the idea of BRNN query in Euclidean distance. When a set of query objects *S* and a set of data objects *P* and a query point *q* (∈ S) are given, BRNN query retrieves all data points in *p*(∈ P) that are nearest to *q* than any other points in *S*. In this figure, p_1 is BRNN of s_1, p_4 is BRNN of s_2 and p_2 and p_3 is BRNN of s_3.

Hereafter, how MRNN query works with Eager algorithm, proposed by Yiu et al. [1] which is followed the lemma 1, is described by Figure 4. Road network nodes are visited from *q*to surrounding nodes in a method similar to that of Dijkstra's algorithm.When the query *q*is on *A* in this figure, node *B* is visited first.Next, at most *k* NNs of B is searched for within the distance Dst=$d_N(B,A)$. This function is called rangeNN(*n*,*q*,*Dst)*. In the above example, *n* is *B* and *q* is *A*. For simplicity, we only consider for one *k*. In the previous query, *C* is found as *B*'s NN. Then, we check whether *C* is included as an RNN of *A*. This check can be done to investigate whether *A* is the NN of *C*.

This function is called verify(*p*,*k*,*q)* and returns true when *q* is the NN of *p*, otherwise, it returns false. In this example, the result of verify(*C*,1,*q)* is true; therefore, *C* is determined as an RNN of *q*. The next visited node is *D*; thus, rangeNN(*D*,*q*,5) is called and *E* is obtained as the NN of *D*. To check whether *E* is a RNN of *q*, verify(*E*,1,*q)* is called; however, false is obtained in this case. Hence, edges beyond *D* are safely pruned. At this time, there is no search path left, therefore, the search process is terminated.

In Yiu's Eager algorithm, two methods, named as verify(*p*,*k*,*q)* and rangeNN(*n*,*q*,*Dst)* are used. For simplicity, these functions are here after denoted as verifyRNN and rangeNN.

The disadvantages intheEager algorithm are summarized as two: (1)a large search area for the verifyRNNandrangeNN functions, (2)a drastic increase in processing time caused by performing rangeNN on every visited node on the road network distance.

To cope with these problems, we propose a method to adapt an IER framework for the verifyRNN and rangeNN methods. Furthermore, we present an efficient method of R*k*NN search to perform MR*k*NNand BR*k*NN queries on the SMPV: (1) to adapt an IER framework for both rangeNN and verifyRNNand (2) to use the Eager algorithm only on the border nodes in the SMPV.

4.1. R*k*NN on SMPV structure

The reason of poor performance in the Eager algorithm is invoking rangeNNat every visited node, and it takes long processing times.In Algorithm 1 and 2 in the proposed method, rangeNN is invoked only on the border nodes of the partitioned graphs to overcome the deficiency in the Eager algorithm.

Algorithm 1, the procedure StartPG is invoked to determine the partitioned graph to which a given query point *q* belongs as expressed in line 2 of Algorithm 1. Let the data point set be *P*. Then in line 4, each element in *P* is checked to determine whether *q* is an R*k*NN of *q* or not. This procedure is the same as in verify(*p*,*k*,*q*) in the Eager algorithm. The verifyRNN searches for the *k*NNs of each *p*∈*P*, and then if *q* is included in the *k*NN set, *p* is determined to be an R*k*NN of *q* and added to the result setin line 5.

Algorithm 1 StartPG

1: **procedure** StartPG(*q*,*PQ*,*R*)
2: *p*g←*determinePG*(*q*)
3: *P*←*f indPOIinPG*(*q*)
4:**for all** *p* ∈ *P***do**
5: **if** verifyRNN(*p*,*k*,*q*) **then** *R p*
⍰add *p*toto result set 6: **end if**
7: **end for**
8: **for all** *b*∈ *BN* **do**
9: PQ.enQueue(<*dN*(*q*,*b*),*b*,*q*,pg>)
10:**end for**
11: **end procedure**

This check needs a wide range search and is not exclusive to only a partitioned graph, hence, IER [5] can be efficiently perform it using SMPV.

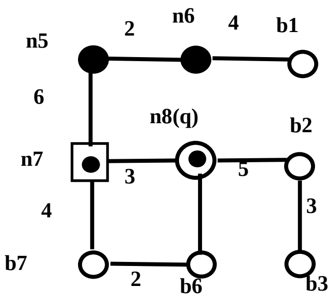

Figure 6.Processing of a cell where q exists

Figure 6 shows the PG_1 as in Figure 2. In this example, a query point *q* is on node n_8. A square overlapped on node $n_7$indicates a data point. For simplicity, the following explanation considers the case for which *k* is one. By searching for the NN of n_7, *q* is obtained as the result. Therefore, $n_7$is an RNN of *q*. Consequently, $n_7$is added to the result set. Next, the search area is enlarged to include the neighboring partitioned graph. For each border node *bi* of this partitioned graph, the distance from *q* to *bi*is obtained by referring to the IBDT of the related partitioned graph. Thereafter, a record is composed and inserted into priority queue PQ. The record is composed as

$$<d,n,p,cid>$$

where*d* is the road network distance between *q* and the border node concerned (*n*), *p* is the previous node on the shortest path from *q* to *n*, and *cid*denotes the partitioned graph ID to which *n* belongs. The first record inserted into PQ is as follows.

$$<d_N(q,bi),bi,q, PG_1>$$

Here, PG_1denotes the partitioned graph in which *q* is included. Line 8 to 10 indicates insertion process into PQ.

Next, the R*k*NN search starts. When a record is dequeued from PQ, the search propagates to the neighboring partitioned graphs. Figure 7 illustrates a partitioned graph PG_1 in which query point *q* is included and PG_2 as its neighboring partitioned graph.

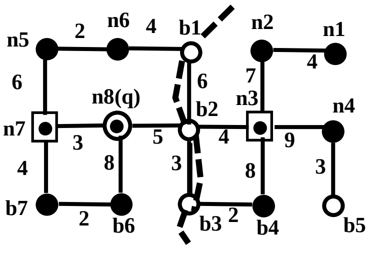

Figure 7.Border node expansion

When a record *r* is dequeued from PQ and *r.n*is the border node b_2, data points in PG_2 are searched. In this partitioned graph, a node $n_3$is included. The *k*NNs of $n_3$are searched, and if *q* is included in the *k*NN set, $n_3$is added to the result set. Otherwise, $n_3$is ignored. This partitioned graph can be visited several times from different border nodes. Therefore, PG_2 is marked as visited to avoid duplicate searches. In the next step, rangeNN is invoked from the border node *bi* to find candidate data points. If the result set is not empty, verifyRNN is invoked to check whether each found data point is truly an R*k*NN of *q*. If the result of verifyRNNis true, the data point is added to the result set. If the size of the result set returned from rangeNN is smaller than *k*, there can exist other R*k*NNs on the path through this node *v.n*, and still cannot prune the search. Therefore, new records from *bi* to the other border nodes in the partitioned graph are created and inserted into PQ.

Algorithm 2 shows the pseudo-code of the proposed method described above. Lines 3 to 12 are similar to the process described by the Eager algorithm. When the record *v* is obtained from PQ, at most number of *k* NNs of the road network node *v.n*are searched and put into *KNN*. For each element *p* of *KNN*, *p* is checked whether *q* is included in its *k*NN. If it is included,*p*is added into the result set R.

Line 13 of Algorithm 2 checks whether thenumber of elements in *KNN* is less than *k*; i.e., the numberofrangeNN resulted data points that are existing in the area centered at *v.n*, and having smaller distance than $d_N(v.n, q)$ is less than *k*. If so, node *v.n*is expanded and the search is continued. Otherwise, no more R*k*NNs exists on the path through *v.n*; therefore, node expansion at *v.n*is not executed.

Algorithm 2 R*k*NN

1: **function**R*k*NN(q)
2: $PQ \leftarrow \emptyset$, $R \leftarrow \emptyset$
3: StartPG(*q*; *PQ*,*R*)
4: **while** PQ not empty **do**
5:v ←*PQ.deQueue*()
6: *CS.add*(v)
7: *KNN* ←rangeNN(v.*n*,*k*,*dN*(v.*n*,*q*),*PQ*)
8: **for all** *p* in KNN **do**
9: **if** verifyRNN(*p*,*k*,*q*) **then**
10: $R \leftarrow R \cup p$
11:**end if**
12: **end for**
13: **if** |*KNN*|<*k* **then**
14: **for all** $b \in BN$ **do**
15:**if** v.cid is visited first time **then**
16: *CP*←*f indPOIinSG*(v.*cid*)
17: **for all** $p \in CP$ **do**
18:**if** verifyRNN(*p*,*k*,*q*) **then**
19: $R \leftarrow R \cup p$
20: **end if**
21: **end for**

```
end if
PQ.enQueue(<dN(q,b),b,p,v.cid>)
end for
end if
end while
return R ← RkNN of q
end function
```

5. Experimental Evaluations

To evaluate our proposed method for R*k*NN comparing to the existing Eager algorithm, several experiments have been done by using the real road network data of Saitama city map whose nodes are 16,284 and links are 24,914. We generated variety ofdensity(D) of data point sets on the road network links by pseudorandom sequences. For instance, *D* = 0.01 means that a data point exists once every 100 links. Both algorithms were implemented in Java and evaluated on a PC with Intel Corei7-4770 CPU (3.4GHz) and 32GB of memory.

Figure 8 and 9 show the processing times of MR*k*NN queries. In the figure 8, the densityof data points isset to 0.01. In this figure, the horizontal axis shows *k*value for MR*k*NNand the vertical axis shows the processing times in seconds to search *k*NNs. As shown in the figure, the processing time of the Eager algorithm sharply increases with *k* because the search area also expands. In contrast, the proposed algorithm linearly increases with *k*.

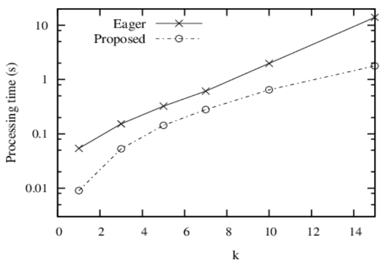

Figure 8.Processing time when *D* is 0.01

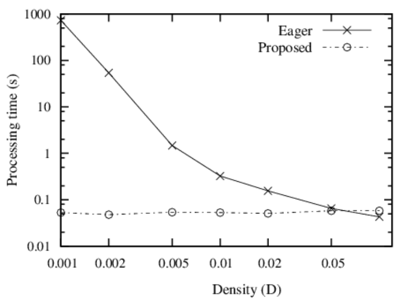

Figure 9. Processing time when *D*varies

Figure 9 shows the processing time on varying the density of data points. In the figure, the horizontal axis shows the density and the vertical axis shows the processing time in secondswhen*k*is 5. The processing time of the Eager algorithm increases sharply when the density is low. On the other hand, the proposed algorithm remains fast even in that case. When the density of data points is high, the Eager algorithm performed well because the size of the search area decreases with the increase in the density. The proposed algorithm shows stable characteristics and independent of the probability.

Figure 10 shows the processing time forBR*k*NNquery.This figure measures the processing time for BR1NN by varyingthe*D* for *S*(query points) set when *D* of *P*(interest points) set is 0.002.In this result, the horizontal axis shows the varied *D* of *S* set and the vertical axis is the processing time. When*D*of*S*setis low, searching in wide range is necessary, and in such case, the Eager algorithm takes long processing time. Conversely, when *D* value increases, the searching area becomes narrow and processing time is faster in Eager algorithm. However, our proposed method showed the stable characteristic and independent of the D for *S* set.

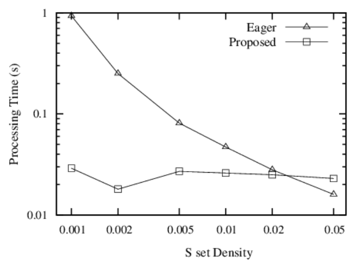

Figure 10. Processing time varying D for S

6. Conclusion

In this paper, we proposed a fast R*k*NN query in road network distance by using the simple materialized path view (SMPV) data. We presented two types of R*k*NNquery,MR*k*NN and BR*k*NN query. With extensive experiments, we showed that the performance of the proposed method comparing with the existing method, Eager algorithm. Especially, the proposed method is 10 to 100 times faster in processing time for both types of R*k*NN query when the number of *k* is large and when the density distribution of points is sparse on a road network. On the other hand, the Eager algorithm has a merit for the very dense distribution of points on road network. Hence, it is considered to refine a new approach by the combination of the strength of our proposed method and the Eager algorithm in order to obtain a more efficient and adaptive query which is not depending on the density distribution of data points on a road network. To advance this concept is for our future work.

Acknowledgments

This study was partially supported by the Japanese Ministry of Education, Science, Sports and Culture (Grant-in-Aid Science Research (C) 24500107).

References

[1] Yiu, M. L., Papadias, D., Mamoulis, N. and Tao, Y.: Reverse Nearest Neighbor in Large Graphs, *IEEETransaction on Knowledge and Data Engineering*, Vol. 18, No. 4, 2006, pp.1-14.

[2] Hlaing, A. T., Htoo, H., Ohsawa, Y., Sonehara, N. and Sakauchi, M.: Shortest Path Finder with Light Materialized Path View for Location Based Services, *Proc. WAIM2013*, Vol. LNCS7923, China, 2013, pp. 229-234.

[3] F. Korn and S. Muthukrishnan: “Influence sets based onreverse nearest neighbor queries”, ACM SIGMOD Record, Vol. 29, 2000, pp. 201-212.

[4] I. Stanoi, D. Agawal and A. E. Abbadi: “Reverse nearest neighbor queries for dynamic databases”, Proc. of 2000 ACM SIGMODWorkshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp. 44-53.

[5] Y. Tao, D. Papadias and X. Lian: “Reverse *k*nn search in arbitrary dimensionality”, Proceedings of the 30th VLDB, Conference, 2004, pp. 744-755.