Versatile Safe-Region Generation Method for Continuous Monitoring of Moving Objects in the Road Network Distance

Yutaka Ohsawa(✉) and Htoo Htoo

Graduate School of Science and Engineering, Saitama University, Saitama, Japan
ohsawa@mail.saitama-u.ac.jp

Abstract. This paper proposes a fast safe-region generation method for several kinds of vicinity queries including distance range queries, set k nearest neighbor (NN) queries, and ordered kNN queries. When a user is driving a car on a road network, he/she wants to know objects located in a vicinity of the car. However, the result is changing according to the movement of the car, and therefore, the up-to-date result is always expected, and requested to the server. On the other hand, frequent requests for updating results to the server cause heavy loading. To cope with this problem efficiently, the idea of safe-region has been proposed. This paper proposes a fast generation method of the safe-region applicable to several types of vicinity queries. Through experimental evaluations, the proposed algorithm achieves a great performance in terms of processing times, and is one or two orders of magnitude faster than existing algorithms.

1 Introduction

With the availability of wireless communication and enormous utility of mobile devices, continuous queries over moving objects have been an explosive increase in the demand for various location based services (LBS) applications. A continuous query continuously monitors over data objects which meets a specific query condition to a query point. Contrary, when a query is answered only one time over static objects, this type of query is known as a snapshot query.

Various algorithms for continuous queries in Euclidean distance have been actively researched in the literature. In practice, moving objects including cars and pedestrian are moving on a road network in real world scenarios, and henceforward, queries based on distance or travel time on road network is more preferable than in Euclidean distance.

Some literature proposed algorithms adaptable to the road network distance, however, they are targeting to an individual type of query, for example, k nearest neighbor (kNN) query, reverse k nearest neighbor (RkNN) query, and distance range query. The aim of this study is to develop a framework adaptable to versatile vicinity queries, including set-kNN, ordered-kNN, RkNN, and distance range query.

H. Gao et al. (Eds.): DASFAA 2016 Workshops, LNCS 9645, pp. 377–392, 2016.
DOI: 10.1007/978-3-319-32055-7_32

In applications for continuous queries, the client-server model is configured in general. In this architecture, when a moving object sends a query to the server, a thread to manage each moving object is initiated at the server side. Then the server sends back its query result to the moving object. The moving object changes the location continuously, and thus, the query result becomes useless according to the location changes. To cope with this problem, three types of update methods have been proposed; (1) update periodically, (2) update by the fixed distance move, (3) set safe-region.

The periodical update and the update by the fixed distance move are apt to overlook the changes of the result or repeat futile queries to get the same result. Hence, safe-region methods have been proposed to solve these problems. The safe-region is an area on the road network in which the query result remains unchanged. As long as a moving object is inside the safe-region, a new query request to the server is not necessary. On the other hand, when the moving object leaves the region, the query result becomes different at the location, and therefore the moving object requests the server to send a new query result and the safe-region.

Figure 1 shows an example of a safe-region of ordered-3NN query at q by thick line. The result of the ordered-3NN of q are data points a, b, and c. In this case, when a moving object leaves the safe-region, it issues a new query to the server, and gets a new ordered-3NN data objects and the safe-region.

Fig. 1. Example of a safe-region

The safe-region approaches have been actively studied as described in the next section, however, these algorithms have been developed for each individual query type. This paper proposes a fast algorithm applicable to versatile vicinity queries in the road network distance. The contributions of the paper are summarized as follow:

- we propose a generation method of versatile safe-region on demand in the road network distance which is applicable to vicinity queries including set-kNN, ordered-kNN, reverse-kNN, and distance range query.
- we evaluate our proposed method comparing to existing works, and show that the proposed method has a great performance in terms of processing time, and is one or two orders of magnitude faster than existing approaches.

The rest of the paper is organized as follow. In Sect. 2, related works of continuous queries are discussed. Section 3 presents the basic principles in safe-region generation for vicinity queries. In Sect. 4 discusses how to improve the efficiency for continuous queries, the generalization of the algorithm for various type of vicinity queries, and the determination of the border points on the road network edge. Section 5 describes evaluations of the performance of the proposed method. Section 6 concludes this paper and describes future works.

2 Related Work

Continuous queries for the moving objects have been actively researched since 2000s. They can be classified into three main categories based on (1) query types, (2) Euclidean distance or road network distance, and (3) mobility nature of queries and data objects.

In the literature, variety of continuous query types have been researched, consisting of range query [1,2], kNN query [3], RNN query [4,5], spatial semi-join query [6], path NN query [7], skyline query [8].

In continuous queries for moving objects, researches have been mainly focused on Euclidean distance in the pioneer studies. However, the movement of cars and humans are constrained on a road network in practice. To the best of our knowledge, Mouratidis et al. [3] first proposed a continuous query method in the road network distance. In their approach, k nearest neighbors are continuously monitored on road network, where the distance between a query and a data object is determined by the length of the shortest path connecting them.

In real world scenarios, continuous queries can be categorized into three groups depend on the mobility of a query and/or data objects [9].

(1) moving query objects querying static data objects.
(2) static query objects querying moving data objects.
(3) moving query objects querying moving data objects.

For an instance in case (1), a user queries a convenience store while driving a car. In case (2), a person queries to get the closest available taxi among taxis running around. Cheema et al. [10] proposed an algorithm for the reverse k nearest neighbor query (RkNN) based on (2). Researches on mobility of both queries and data objects (case (3)) have been introduced by Stojanovic et al. [9], Cheema et al. [11] and Liu et al. [12]. These types of queries are necessary, for example, a car queries to nearest rival cars within a certain region while driving in a car race.

Continuous queries are generally based on the client-server model, and the task of a server is to continuously compute and update the result of each query according to the location changes of the moving objects. Consequently, queries are repeated periodically or by moving a certain distance. However, if the frequency of updates becomes high, the performance in monitoring declines.

To overcome overloads at the server side, Prabhakar et al. [2] proposed a safe-region method. When a moving object queries for kNN or range query, the server generates a safe-region in which the query result remains unchanged. By the time the moving object leaves the safe-region, a new query result and the safe-region are requested to the server. Thereafter, various continuous query methods based on safe-regions have been proposed focusing on Euclidean distance. Yiu et al. [13] proposed efficient algorithms for RkNN queries in road network distance. Among them, the Eager algorithm can be applied to safe-region generations for static RkNN queries.

Alternatively, Cheema et al. [14] proposed an efficient and effective monitoring technique based on the safe-region for range queries: they used the term

of safe-zone instead of safe-region. They also proposed safe-region generation method for continuous RkNN queries. Although safe-region generation methods have been actively researched, these algorithms were proposed for an individual query type. Moreover, these algorithms are based on a similar methodology in region expansions. In region expansions, the region is gradually expanded verifying the query condition from the query point. The most time consuming step is the verification for the query condition at each network node.

This paper first proposes a safe-region generation method applicable to versatile vicinity queries; including set-kNN, ordered-kNN, reverse-kNN, and distance range queries. Secondly, the performance efficiency is improved in the verification process, which checks whether the query condition is satisfied or not, by applying incremental Euclidean restriction (IER) [15] and the idea of the SSMTA* algorithm [16]. As the result, the method presented in this paper improves the processing time to generate safe-region for versatile vicinity queries by one or two orders of magnitude.

3 Basic Principles in Safe-Region Generation

3.1 Safe-Region for Vicinity Queries

First, we give definitions of the safe-regions for three types of vicinity queries related to nearest neighbor queries.

Definition 1 (Safe-region for Set-kNN Query). *Safe-region is a region where kNN queries invoked anywhere in the region give the same set of data points, ignoring the order in distances.*

Definition 2 (Safe-region for Ordered-kNN Query). *Similar to set-kNN query, however, the order of the distances in the kNN result is considered.*

Definition 3 (Safe-region for RkNN Query). *Safe-region is a region where kNN query result invoked anywhere in the region always contains a specified given data point.*

For an example, in Fig. 2(a), $p1 \sim p4$ belong to a data objects set P, and $a \sim d$ are query points. Here, we deal with 2NN of each point. For a, the 2NN is $p1$ and $p2$, and for b, they are $p2$ and $p1$. Therefore, when the order of 2NN is considered for the ordered-kNN query, the result of their 2NN are different. However, the result set of the 2NN queries are the same with $\{p1, p2\}$ in any order for the set-kNN query. This is the difference between the set-kNN and the ordered-kNN queries. Furthermore, the 2NN of c is $p4$ and $p1$, as the result, all 2NN of a, b and c contain $p1$. As a consequence, $p1$ is included in 2NN results invoked from a, b and c. Therefore, when $p1$ is specified as a query point, 2NN queries invoked at a, b, and c include $p1$ in their results. Therefore, the safe-region of R2NN of $p1$ includes these three data points. In set-kNN and ordered-kNN queries, the space is partitioned into non-overlapping regions. Contrary, the space is divided into mutually overlapping regions in RkNN query (except when k is one), and

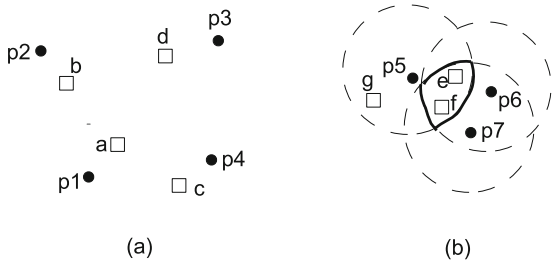

Fig. 2. Examples of vicinity queries

therefore, generated safe region for RkNN is larger than in the ordered and set-kNN queries. Hence, to specify a data object as the query point is necessary for RkNN query.

Definition 4 (Safe-region for Distance Range Query). *Safe-region is a region where distance range query, with a given distance D and a query point, invoked anywhere in the region contain the same set of data points.*

Figure 2(b) shows an example of the distance range query. In the figure, e, f and g are query points. $p5$, $p6$ and $p7$ are data objects and circles in dotted line show the areas centered at each data object and the radiuses are D. The area shown by bold lines shows the region in which the distance range query result is $p5$, $p6$ and $p7$. The objects e and f are included in this area, therefore, the distance range query results invoked from these two points give the same result $\{p5, p6, p7\}$, and the region in which these two points lie is the safe-region of the distance range query.

Definition 5 (Safe-region for Vicinity Queries). *A query on a road network to find the objects in P whose result is the same. This type of queries includes set-kNN, ordered-kNN, RkNN and distance range query. Especially, a safe region for set-kNN is denoted by SR_s, for ordered-kNN by SR_o, for RkNN by SR_r, and for distance range query by SR_d.*

For the distance range query SR_d, its query condition is considered on a given distance, and different from other nearest neighbor queries. Hence, the relationship among vicinity queries based on kNN queries satisfies the following relationship:

$$SR_o \subseteq SR_s \subseteq SR_r$$

3.2 Basic Principle of the Proposed Method

In this paper, a large road network is considered and modeled as a directed graph $G(V, E, W)$, where V is a set of nodes (intersections), E is a set of edges (road segments), and W is a set of edge weights and $w(\in W) \geq 0$ stands. w is assumed as the length of edge in the rest of the paper.

We define a safe-region (SR) as a region that gives the same query result. The generation of SR is emerged based on Voronoi region. Hence, SR for 1NN, in other words, order-1 Voronoi region (VR) is discussed in this section. The order-1 Voronoi region is the simplest SR. If Voronoi region in the road network distance is to be considered, VR(q) is the set of road segment in which $q(\in P)$ lies as the nearest object. An edge on G may belong to plural VRs, and the edge is exactly partitioned into plural regions to be belonged to respective VRs.

Figure 3 illustrates a road network. In this figure, white circles are road network nodes, black circles are data objects in P and the numbers attached along road segments are distances. The area bounded by the symbol × shows the order-1 Voronoi region in which q is included as the nearest object.

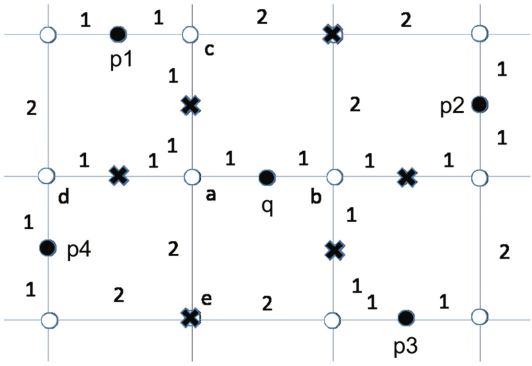

Fig. 3. The order-1 VR

The algorithm described in the next subsection can be applied to several types of vicinity queries. The basic principle how to generate SR for these queries is followed by two steps below:

(1) Gradually expanding the search region from a query point q to adjacent nodes as the similar way in Dijkstra's algorithm.
(2) At every visited node in step (1), verifying whether expanded nodes meet the query condition or not. If the result of verification is true, the node is expanded. Contrary, if the result is false, the node is not expanded anymore.

In the above step (1), a best-first search is applied from a query point to a current noticed node referring to a priority queue (PQ). In the step (2) after verifying whether the current node meets the query condition, all adjacent nodes to the current node are inserted into PQ. Then, the further search is proceeded from these nodes ahead. Contrary, if the query condition is not satisfied at the current node, node expansions from this node ahead are terminated. The query condition differs from each query type. The detail is described in Sect. 4.2.

3.3 Basic Method

In this subsection, an algorithm how to generate the safe region is described. The process in this algorithm is to generate a region by expanding the search area gradually while the query condition is satisfied. The search starts at q, and it is controlled by a best-first search using a priority queue (PQ). The record format in PQ is as follow:

$$< cost, n, p, \ell(p, n) >$$

Here, n represents a current node, *cost* is the cost (the road network distance from q to n), p is the previously visited node to n, and $\ell(p, n)$ is the road segment between p–n. The cost is assumed in the road network distance, and the length of $\ell(p, n)$ is expressed as $d_N(p, n)$ for the road network distance.

At first, for two end nodes of the road segment on which q exists, the following records are inserted into PQ.

$$< 1, a, q, \ell(a, q) >, < 1, b, q, \ell(b, q) >$$

Then, the record with the minimum cost is dequeued from PQ. Thus, $< 1, a, q, \ell(a, q) >$ is dequeued from PQ. Moreover, the nearest neighbor (NN) of a is searched and checked whether the NN is q or not. If q is the NN of a, a also lies in SR(q) where q is a generator of SR.

In the NN search, the incremental Euclidean restriction (IER) [15] framework is used to find the NN faster. In IER framework, the NN candidates are incrementally searched in Euclidean distance, and verified these candidates in road network distances by A* algorithm. The verification process invokes next NN search while the Euclidean distance of the next NN candidate is smaller than or equal to the road network distance to the current NN candidate.

To expand the search range, records for adjacent nodes to a, those are c, d and e in Fig. 3, are enqueued into PQ.

$$< 3, c, a, \ell(c, a) >, < 3, d, a, \ell(d, a) >, < 3, e, a, \ell(e, a) >$$

In the next time, $< 1, b, q, \ell(b, q) >$ is dequeued from PQ and the similar process is performed at the node b.

Then, let $< 3, c, a, \ell(c, a) >$ be the next dequeued record from PQ. Similarly, NN of c is checked whether q is NN of c. In this case, the result is false, therefore c does not lie in $SR(q)$. At this point, a border point is determined on the road segment $\ell(c, a)$. The same process is performed to the next dequeued record $< 3, d, a, \ell(d, a) >$. When the record $< 3, e, a, \ell(e, a) >$ is dequeued, e can be considered as a border point because e is equidistant from q, p_3 and p_4. By repeating the process, SR(q) where q is the generator can be generated.

Algorithm 1 shows the above process in pseudocode. The PQ in lines 2 and 3 is the priority queue (heap) to control the region expansion, and in these lines, initial records at q are inserted into PQ. In line 4, a closed set (CS) is prepared for once checked road segments to avoid duplicated checks. R in line 5 is the result set of road segments included in the safe-region. In line 6, initialSet function is called to search NN of q and the result is assigned into T. The function initialSet is differently implemented depending on the query type. In the case of 1NN query, the 1NN of q is set to T (in this case, only one data object). The detail of this function for general query types is explained in Sect. 4.2.

Line 7 to 22 perform the following process. Initially, a record with the minimum cost is dequeued from PQ, and the road segment of the record $r.\ell$ is checked whether it is already registered in the CS. If $r.\ell$ is in the CS, it means that the segment has already been checked, and the rest steps are skipped. In line 12, $r.\ell$ is added into the closed set. In line 13, the current node $r.n$ is checked whether

the node meets the query condition. As in the example of SR(q), the query condition is that NNs of current node $r.n$ are same as objects in T. If the result of VERIFY is true, the node ($r.n$) is expanded. Hence, adjacent road segments of $r.n$ are searched by referring to the adjacency list, and records for all adjacent road segments are created. Then, these records are enqueued into PQ. Moreover, the whole edge ($r.\ell$) is added into the result set R as shown in line 18. However, if the VERIFY result is false, the function ADDWITHCHECK calculates the part of the edge where the verify condition is still satisfied, and the result part is inserted into R. The detail, how to determine the part of the edge, is discussed in Sect. 4.3.

Algorithm 1. Safe-region generation: SRG

```
1: function SRG(q)
2:     PQ.enqueue(< d_N(n_1, q), n_1, q, ℓ(n_1, q) >)
3:     PQ.enqueue(< d_N(n_2, q), n_2, q, ℓ(n_2, q) >)
4:     CS ← ∅
5:     R ← ∅
6:     T ← INITIALSET(q)
7:     while PQ.size() > 0 do
8:         r ← PQ.deleteMin()
9:         if CS contains r then
10:            continue;
11:        end if
12:        CS ← CS ∪ r.ℓ
13:        if VERIFY(r.n,T) then
14:            ns ← AdjacentNode(r.n)
15:            for all n ∈ ns do
16:                PQ ← PQ∪ < r.d + d_N(r.n, n), n, r.n, r.ℓ >
17:            end for
18:            R ← R ∪ ADD(r.ℓ)
19:        else
20:            R ← R ∪ ADDWITHCHECK(r.ℓ, q, T)
21:        end if
22:    end while
23:    return R
24: end function
```

4 Improving Efficiency and Generalization

4.1 Improvement in the Processing Time

The most time consuming step in Algorithm 1 is at the VERIFY procedure called for every node in the region expansion. For a visiting node n, kNN candidates of n are searched in Euclidean distance, and these candidates are verified in the road network distance. To verify in the road network distance, the A* algorithm can be applied. However, in general SR, at least k number of objects are searched

as targets at every node n. In such condition, even if A* algorithm is fast for a single query, the total processing time becomes long due to repeated searching in adjacent regions.

To improve in the efficiency in terms of processing time for the distance calculation, the idea of the single source multi- targets A* (SSMTA*) [16] algorithm has been applied to the proposed method. The original SSMTA* algorithm concurrently finds the shortest paths from a source node to multiple target nodes efficiently. Contrary, when it is applied to SR generation, the target points are changing sequentially.

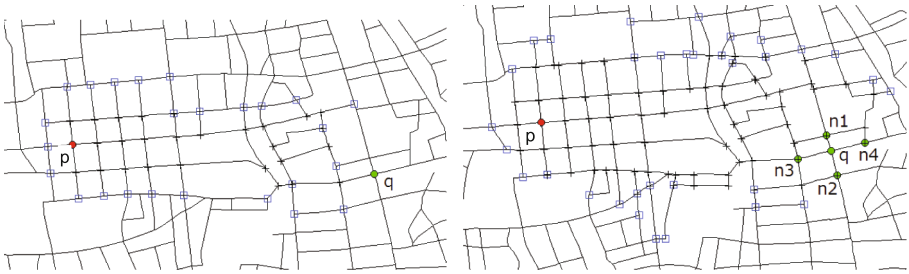

Fig. 4. The distance calculation by SSMTA* algorithm

In Fig. 4(a), q is a query point in SR, and p is a kNN candidate searched in Euclidean distance. The A* algorithm is applied to find the road network distance from p to q. In the A* algorithm, a priority queue PQA and a closed set CSA are used. The record format in PQA and CSA is the following (these are different from CS and PQ introduced in Sect. 3).

$$< c, v, d > \tag{1}$$

Here, v is a current node in A* search, d shows the distance on a road network between p and v. The first item c is the lower bound distance between p and q, that is $c = d_N(p, v) + d_E(v, q)$ where $d_N(p, v)$ is the road network distance between p and v, and $d_E(v, q)$ is the Euclidean distance between v and q.

The priority queue PQA stores these records and returns a record in ascending order of c value. The current nodes (v) in PQA are called wave-front, and their distances from p have not been determined yet. Once a record is dequeued from PQA, it is registered into CSA. Since the distance d in a record in CSA has already been determined, it shows the shortest path length between p and v. The symbols □ in Fig.4 show the nodes in PQA, and the symbols + show the nodes in CSA.

Figure 4(b) shows the region expansion from q to neighboring nodes. In this figure, the search region in Algorithm 1 is enlarged from q to $n1 \sim n4$ gradually. Every time a new neighboring node is investigated, the function VERIFY needs to

calculate the road network distance from p to the node. If A* algorithm is used in this check, almost the same road network nodes are repeatedly processed. To improve the efficiency of the verification process, we reuse the contents of PQA and CSA in the verification of the neighboring nodes.

When a new data object p' becomes a candidate of kNN of a node (in the first step, it is q), the distance between p' and q is obtained by applying A* algorithm. And then, the contents in PQA and CSA are kept for the next search. When a neighbor node (n) becomes a target, the distance between q and n is obtained by one of the following two cases.

case 1 If n has already been in CSA, the distance between q and n can be obtained by referring to d value in the record, that is $n.d$.
case 2 Otherwise, recalculate c value of all records in PQA for a new target point n, and then resume the search by A* algorithm.

When n has been included in CSA, we can obtain $d_N(p, n)$ by case 1. In this case, the search area is not expanded at all. Otherwise, the c values in PQA are recalculated by the equation $c = d_N(p, v) + d_E(v, n)$. This process is necessary because the target node (n) is changed. By referring to updated PQA, the distance search targeting to n is started again. The basic A* algorithm (called pair-wise A* algorithm) finds the shortest path from p to n repeatedly every time the target point is changed. Comparing to it, it is realized that the processing time can be considerably reduced in the improved method. Moreover, the update process for all records in PQA is taken place in the memory and it does not take long processing time.

Figure 4(b) shows the wave-front of PQA and contents of CSA for searching the shortest paths targeted to 5 adjacent nodes (q and $n1 \sim n4$). Comparing to Fig. 4(a), the region of expanded nodes in PQA and CSA is larger, however, the total number of nodes are smaller than repeatedly invoking pair-wise A* algorithm. By pair-wise A* algorithm, the similar node expansion to Fig. 4(a) must be repeated five times, and then, the total number of nodes becomes about five times for Fig. 4(a).

4.2 Generalized SR

The algorithm for SRG shown in Algorithm 1 is applicable to a variety of kNN queries including set-kNN, ordered-kNN and RkNN. To adapt these queries, three functions, INITIALSET(q), VERIFY and ADDWITHCHECK are needed to prepare for an individual query. Among them, ADDWITHCHECK is described in Sect. 4.3.

In set-kNN query and ordered-kNN query, INITIALSET(q) returns kNN data objects to q as the result. In set-kNN query, VERIFY(n,T) returns true if kNN results at node n are exactly the same (order omitted) as the objects in the set T. In ordered-kNN query, VERIFY returns ture when kNN results at node n is exactly the same as the objects in T in a sorted order.

In RkNN query, 1NN to q in data objects is searched by the result of INITIALSET(q), and returns the data object (let s_q be a specified query point). This means that T holds only one data point s_q. The VERIFY(n,T) returns true if s_q is included in kNN results of n.

The distance range query is also included as a variation of the vicinity query. In this query, INITIALSET(q) searches the data objects located in the range whose distance from q is less than or equal to D (the radius of the range), and they are set to T. In VERIFY, the distances from the current node n to each object in T are investigated. If all distances do not exceed D and do not include any other data object except objects in T, it returns true. Otherwise, it returns false.

4.3 The Borders Determination on Edges

The safe-region is a collection of road network edge segment on which the query condition is satisfied at any part. In Sect. 4.1, we described that the verification of the query condition was checked at road network nodes. If the query condition is satisfied at a node, it is also satisfied on the whole edge ended at the node. Therefore, the whole edge is added into the safe-region by ADD in Algorithm 1. On the contrary, when the query condition is not satisfied at a node, the border of the safe-region exists on the edge. How to determine the border points on the road network have been studied by Cho et al. [17,18].

Figure 5(a) shows a part of a road network, and in this example VERIFY returns 'true' at node A, but it returns 'false' at node B. Then, a border of the safe-region exists on the edge labeled LinkL (shown by bold line). In this figure, 'd', 'e', and 'g' are data objects.

Figure 5(b) shows the distance between each object and node A (left vertical axis), and between each object and node B (right vertical axis). The horizontal axis shows the normalized position on the edge. The lines in this figure show the distance change from each data object. The absolute value of all gradient of lines are the same. If there are plural objects existed on the left side of A, plural lines with the same gradient but different intercept appear. The similar situation appears for the right side of B. If there are plural objects on the edge having 'e', lines similar with 'e' but having different peak positions and intercepts appear. The border point on LinkL can be determined easily to find the nearest position from A and beginning at the position where the query condition is not satisfied.

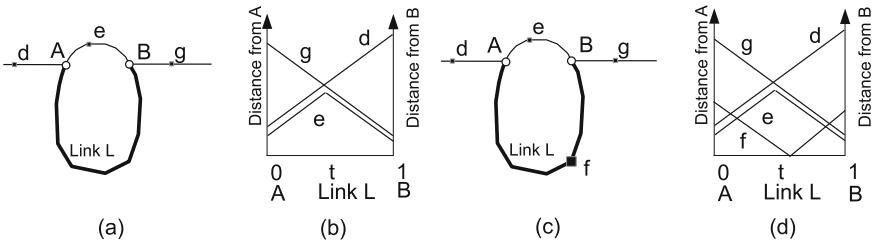

Fig. 5. The borders determination on edges

Figure 5(c) shows another case, a data object 'f' is on LinkL. In this case, a new line type appears in Fig. 5(d), the distance between A and 'f' decreases according to the move toward B, and it becomes zero at 'f', and then it increases according to the move toward B. The determination of the border position becomes complex a little, however, the method how to determine is the same with the case Fig. 5(a) and (b).

5 Experimental Results

This section presents evaluations of the proposed method comparing to existing works. Algorithms were implemented by Java language. The computer used for this evaluation was Intel Core i7 4770 CPU (3.4 GHz). Table 1 shows the road networks used in this experiment. MapA is a road network of the center part of a city, and MapB includes the center of a city and rural area. Data points to be searched were generated by pseudo-random sequence on the road network edges with various densities. For example, the density of 0.001 means a data point exists once 1,000 road edges. For the moving paths of an object, we used both real paths and randomly generated paths for moving objects in experiments. To generate a path on a road network, we randomly set a start point s and a destination point e, and a moving object was started a move from s to e via the shortest route. When the moving object arrived at e, a new destination point e was set and the moving object continuously moved to e from the current location. By repeating this process, paths for moving objects were generated. Besides, we prepared 100 real paths. It took about 30 min for the moving object to move on each path.

Table 1. Road maps

Map-name	No. of node	No. of link	Area-size
MapA	16,284	24.914	168 km^2
MapB	109,373	81,233	284 km^2

Figure 6(a) shows the processing time to generate a safe-region for set-kNN queries. In this figure, 'Basic' shows the processing time of the basic algorithm described in Sect. 4.1 (it is an existing algorithm) and 'Prop' shows the processing time of the proposed method. 'A' and 'B' in the parentheses correspond to 'MapA' and 'MapB' respectively. The horizontal axis shows k, the number of nearest neighbors to be searched. The density of the data points were set to 0.005. As shown in this figure, the proposed method requires less than one-tenth to the basic method. Figure 6(b) shows the processing time when the density of the data points was varied. The value of k was fixed to 5 in this experiment.

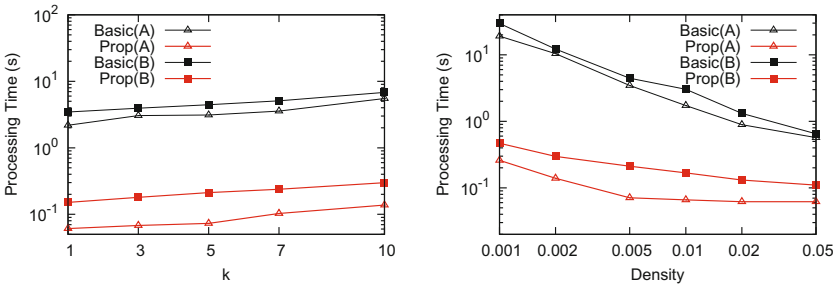

Fig. 6. The processing time of a safe-region for set-kNN

Figure 7 shows the processing time to generate a safe-region for ordered-kNN queries. The size of the safe-region in ordered-kNN queries becomes smaller than in set-kNN queries, because the order of the distances is also considered in ordered-kNN queries. Accordingly, the processing time in both proposed method and the basic method becomes faster in this case. However, the proposed algorithm still outperforms the basic algorithm.

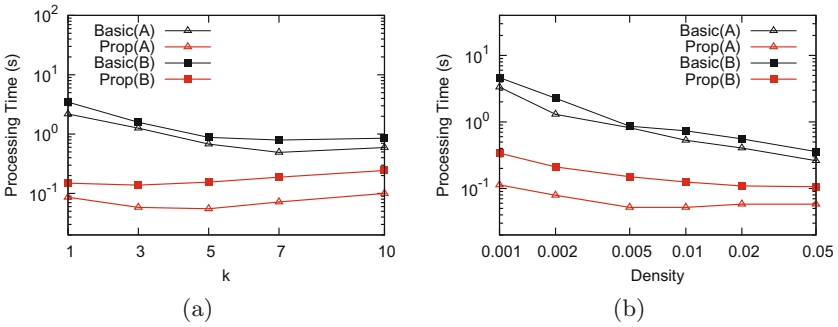

Fig. 7. The processing time of a safe-region for ordered-kNN

Figure 8 compares the processing time for reverse-kNN queries. In this experiment, the nearest-neighbor (p) to the current position of the moving object is first searched, and then the region where p is included in the kNN set is retrieved. This means that p is always included in the kNN of the moving object which moves inside the safe-region. The size of the safe-region of reverse-kNN becomes the biggest among three types of nearest neighbor queries. Therefore, the processing time is also longer than set-kNN and ordered-kNN queries. However, the processing time of the proposed algorithm is one to two orders of magnitude shorter than the basic algorithm.

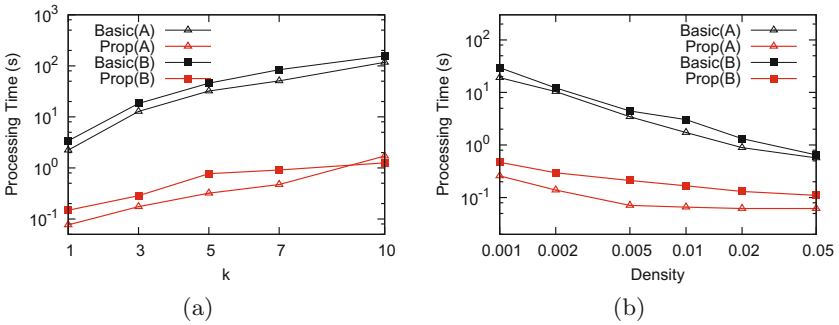

Fig. 8. The processing time of a safe-region for reverse-kNN

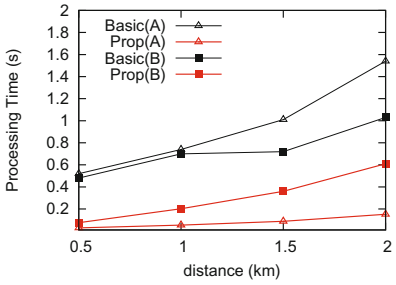

Fig. 9. Distance range query

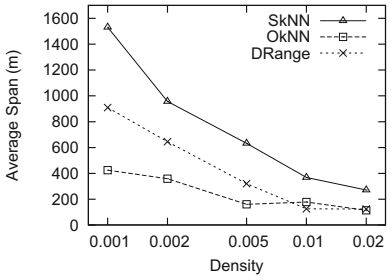

Fig. 10. Average distance between queries

Figure 9 shows the processing time in generating the safe-region for distance range query when distances D vary. In this experiment, the density of data points was fixed to 0.005. When D is larger, the processing time becomes longer. However, the proposed algorithm is also faster than the basic algorithm.

In Fig. 10, we measure the communication cost to the server in terms of traveling distance by a moving object. When a moving object is moving along the route and it reaches the end of the safe-region, a new query result and its safe-region are provided by the server. The frequency of query requests depends on the traveling distance across the safe-region. In this figure, the vertical axis shows that the average traveling distance of a moving object within a safe-region, and the horizontal axis shows the density of data points for three types of queries. In this figure, the value of k is set to 5 for SkNN (set-kNN) and OkNN (ordered-kNN) queries, and D is 1.5 km for distance range query. As shown in this figure, the average traveling distance decreases according to the density increase. When the data density increases, the distance within the safe-region becomes shorter and frequent generation of new safe regions are requested to the server. Consequently, the communication cost becomes higher in such situation.

6 Conclusion

In this paper, a versatile safe-region generation method for continuous queries over moving objects in road network distance is proposed. In the safe-region generation process, repeated expansions over same road network nodes are avoided in the proposed method, and it improves the efficiency. Moreover, in evaluations, the proposed method is applied to continuous vicinity queries including set-kNN query, ordered-kNN query, reverse-kNN query, and distance range query. Comparing to existing works, our proposed method has a great efficiency in terms of processing time, and shows that two orders of magnitude faster than existing approaches especially when data objects are sparsely distributed. To apply the proposed method to more complicated spatial queries is future works.

Acknowledgments. The present study was partially supported by the Japanese Ministry of Education, Science, Sports and Culture (Grant-in-Aid Scientific Research (C) 15K00147).

References

1. Gedik, B., Liu, L.: MobiEyes: distributed processing of continuously moving queries on moving objects in a mobile system. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 67–87. Springer, Heidelberg (2004)
2. Prabhakar, S., Xia, Y., Kalashnikov, D., Aref, W., Hambrush, S.: Query indexing and velocity constrained indexing: scalable techniques for continuous queries on moving objects. IEEE Trans. Comput. **51**(10), 1124–1140 (2002)
3. Mouratidis, K., Yiu, M.L., Papadias, D., Mamoulis, N.: Continuous nearest neighbor monitoring in road networks. In: Proceedings of the 32nd VLDB, pp. 43–54 (2006)
4. Bentis, R., Jensen, C.S., Karčlauskas, G., Šaltenis, S.: Nearest and reverse nearest neighbor queries for moving objects. VLDB J. **15**(3), 229–250 (2006)
5. Xia, T., Zhang, D.: Continuous reverse nearest neighbor monitoring. In: Proceeding of the 22nd International Conference on Data Engineering, p. 77 (2006)
6. Iwerks, G.S., Samet, H., Smith, K.P.: Maintenance of spatial semijoin queries on moving points. In: Proceedings of VLDB (2004)
7. Chen, Z., Shen, H.T., Zhou, X., Yu, J.X.: Monitoring path nearest neighbor in road networks. In: SIGMOD 2009, pp. 591–602 (2009)
8. Huang, Y.K., Chang, C.H., Lee, C.: Continuous distance-based skyline queries in road networks. Inf. Syst. **37**, 611–633 (2012)
9. Stojanovic, D., Papadopoulos, A.N., Predic, B., Djordjevic-Kajan, S., Nanopoulos, A.: Continuous range monitoring of mobile objects in road network. Data Knowl. Eng. **64**, 77–100 (2007)
10. Cheema, M.A., Lin, X., Zhang, W., Mhang, Y.: Influence zone: efficiently processing reverse k nearest neighbors queries. In: Proceeding ICDE, pp. 577–588 (2011)
11. Cheema, M.A., Zhang, W., Lin, X., Zhang, Y., Li, X.: Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks. VLDB J. **21**, 69–95 (2012)

12. Liu, F., Do, T.T., Hua, K.A.: Dynamic range query in spatial network environments. In: Bressan, S., Küng, J., Wagner, R. (eds.) DEXA 2006. LNCS, vol. 4080, pp. 254–265. Springer, Heidelberg (2006)
13. Yiu, M.L., Papadias, D., Mamoulis, N., Tao, Y.: Reverse nearest neighbor in large graphs. IEEE Trans. Knowl. Data Eng. **18**(4), 1–14 (2006)
14. Cheema, M.A., Brankovic, L., Lin, X., Zhang, W., Wang, W.: Continuous monitoring of distance based range queries. IEEE Trans. Knowl. Data Eng. **23**, 1182–1199 (2011)
15. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: Proceedings of 29th VLDB, pp. 790–801 (2003)
16. Htoo, H., Ohsawa, Y., Sonehara, N., Sakauchi, M.: Incremental single-source multi target A* algorithm for LBS based on road network distance. IEICE Trans. Inf. Syst. **E96–D**(5), 1043–1052 (2013)
17. Cho, H.J., Kwon, S.J., Chung, T.S.: A safe exit algorithm for continuous nearest neighbor monitoring in road networks. Mobile Inf. Syst. **9**, 37–53 (2013)
18. Cho, H.J., Chung, C.W.: An efficient and scalable approach to CNN queries in a road network. In: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 805–876 (2005)