Stepwise Optimisation Method for *k*-CNN Search

Jun FENG†, Naoto MUKAI††, and Toyohide WATANABE††

† Department of Information Engineering,
Graduate School of Engineering, Nagoya University.
†† Department of Systems and Social Informatics,
Graduate School of Information Science, Nagoya University.
Furo-cho, Chikusa-ku, Nagoya, Aichi 464-8603, JAPAN

Abstract The problem of *k*-CNN search along a specific route on road network is to find out *k* nearest neighbor (*k*-NN) objects for any place on the route. *k* nearest neighbors are selected based on the path length from the route to the objects, and the continuous search for all the points on the route should be considered. A *k*-CNN search method is proposed by using an incremental *k*-NN search based on road network. The method is an extension of CNN search method proposed by adding new data structure – a fixed length queue for recording up-to-now intermediate results. Because the search regions can be reduced stepwise by the intermediate results, our *k*-CNN search is efficient.

Key words Geographical Information System, path searching, spatial index, road network

1. Introduction

The problem of *k* continuous nearest neighbors (*k*-CNN) search along a specific route on road network is to find out *k* nearest neighbor objects for any place on the route. It is an instance of spatial distance semi-join problem. The spatial distance join associates one or more sets of spatial object by distances between them. The distance semi-join is a useful special case of the distance join. It finds the nearest object in one spatial dataset *T* for each object in another spatial dataset *S* [1]. A distance is usually defined in terms of spatial attributes, but may also be defined in many different ways according to various application-specific requirements. In GIS and ITS applications, for example, other metrics such as the shortest path can be used to measure a distance between two places on a road network.

A spatial join algorithm typically contains two steps, *filter* and *refinement*, as proposed in [2]. In the filter step, MBR approximations are used to find pairs of potentially intersected spatial objects. Then, in the refinement step, it is guaranteed that all the qualified (i.e., actually intersected) pairs can be produced from the results generated in the filter step. In contrast, it is completely unreasonable to process the *k*-CNN search on road network in two separate filter and refinement steps, because of the fact that a filtering process is based on MBR approximation. The distance order of object pairs measured by MBR approximation does not reflect a true order based on actual representations [3]. This is because, for any two pairs of spatial objects $< s_1, t_1 >$ and $< s_2, t_2 >$ ($s_i \in S$ and $t_i \in T$), the fact that

$$dist(MBR(s_1), MBR(t_1)) \leqq dist(MBR(s_2), MBR(t_2))$$

does not necessarily imply that

$$dist(s_1, t_1) \leqq dist(s_2, t_2).$$

Here, $dist(s_i, t_j)$ is a distance between two spatial objects.

However, based on the properties of MBR approximation, there is

$$dist(MBR(s_1), MBR(t_1)) \leqq dist(s_1, t_1).$$

Therefore, if the longest distance between the first *k* pairs of objects can be predicted, spatial join can be done inside a proper region limited by the distance, and MBR approximations can be used to attain an efficient processing. The distance is a cutoff distance that is determined by *k* and the spatial attribute values of two datasets *S* and *T*.

In this paper, we propose new strategies for efficiently processing *k*-CNN search on road network based on our previous CNN search method [4]. The main contributions of the proposed solutions are:

- We provide an approach for estimating the cutoff distance for *k*-CNN search on road network. This estimated distance allows the algorithms of *k*-NN search for any point on the predefined route to avoid a *slow start* problem, which may cause a substantial delay in the query processing.
- Adaptive algorithms are proposed to process *k*-CNN search in a way that the *k*-NN's are returned with an incremental precision. This algorithm adopts a fixed length queue for recording cutoff distance and the first *k* candidates in the search process.

The rest of this paper is organized as follows. Section 2 surveys the related work on processing CNN and *k*-NN queries. Section 3 depicts preliminaries of path on on road network; Section 4 describes our approaches. Section 5 introduces the algorithms for *k*-CNN search on road network. The conclusion is drawn in Section 6.

2. Related Work

2.1 CNN search

The existing work for CNN search is almost presented from the computational geometry perspective [5] [6] [7]. To the best of our knowledge, the latest work dealing with CNN queries is given in [5]. CNN search for line segments was effective, based on the straight-line distance between objects. The same effect can be found in [7], which only finds the single NN for a whole line segments.

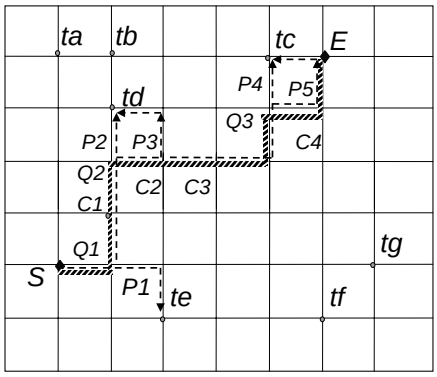

Figure 1 Road network, specific route [*S*, *E*], and target objects.

We have proposed a method [8] to solve the problem based on common situations in GIS: the distance from a point on the route to a target place should be decided by the path length or travel cost of them; and the target objects and the road network are managed in GIS datasets, respectively. Consider the example given in Figure 1, where the specific route is $[S, E]$, and the target object set is $\{t_a, t_b, t_c, t_d, t_e, t_f, t_g\}$. The output of the query is $\{< t_e, [S, Q_1], P_1 >, < t_d, [Q_1, Q_2], P_2 >, < t_d, [Q_2, C_2], P_3 >, < t_c, [C_2, Q_3], P_4 >\}, < t_c, [Q_3, E], P_5 >\}$: the target object t_e is NN for the interval (subroute) $[S, Q_1]$, and the shortest path from the subroute to t_e is P_1; t_d is NN for the subroute $[Q_1, Q_2]$ with the shortest path P_2; t_d is also NN for the subroute $[Q_2, C_2]$ with the shortest path P_3; t_c is that for the subroutes $[C_2, Q_3]$ and $[Q_3, E]$ with the shortest paths P_4 and P_5, respectively.

By selecting computation points heuristically (e.g., $\{S, C_1, C_2, C_3, C_4\}$ in the previous example) and initializing NN search region for these computation points, our CNN search finds the target objects with the shortest path length from all the points on the route effectively.

However, our method only supports to find the nearest neighbor for any point on the predefined route, and does not assure that the candidates for *k*-NN queries can be generated in the search process.

2.2 *k*-NN search

Researches on *k*-NN search have been conducted from the viewpoint of incremental spatial join [3], [9], [10] and distance browsing [1], [11].

In [3], a *k-distance* join algorithm that used spatial indexes such as R-trees was proposed. Bi-directional node expansion and plane-sweeping techniques were used to prune distance pairs, and the plane-sweeping is further optimized by strategies for selecting a sweeping axis and direction. During top-down traversals of R-tree indexes, they store examined node pairs in a priority queue (main queue), where the node pairs are kept in an increasing order of distances. By using the main queue, a spatial distance join query is processed incrementally. Suppose a maximum of *k-nearest* pairs of objects are to be retrieved by a query, any pairs of nodes (and any pairs of their entries) whose distance is greater than all of the *k* candidate pairs cannot be qualified as a query result. Thus, they used another priority queue to store the *k* minimum distances and use the queue to avoid inserting unqualified pairs into the main queue during the node expansions. However, for the semi-join query based on road network (i.e., the *k*-CNN query on road network), the distance function used in the spatial index structure is different from that for computing the distance between objects (the shortest path from a source place to a target place), and the two queues used in their methods cannot be used directly.

In [1] and [11], incremental algorithms were presented to compute the distance join and distance semi-join in the sense that the pairs resulting from the corresponding operation are reported one by one. This enables the query processor to use the algorithms in a pipelined fashion. The algorithms aim to deliver results as soon as possible. Their algorithm is suitable for any spatial data structure based on a hierarchical decomposition. The distance functions are all based on a distance metric for points, $dis(s, t)$, such as the Chessboard, Manhattan or Euler metric. The algorithm functions correctly as long as the distance functions are "consistent". Informally, "consistent" means that no pair can have a smaller distance than a pair that gives rise to it during the processing of the algorithm. For example, if s and t are objects in S and T, respectively, and n is a leaf node which contains S, then they must have $dis(s,t) \geq dis(n,t)$. If the distance functions are all based on the same metric, this condition will hold due to the property of triangle inequality. However, the distance defined on road network is the shortest path from a source place to a target place. When the target objects and road network are managed by different spatial data structures, respectively, to compute the distance on road network should first merge the two data structures. The merging operation is a computational-intensive process.

3. Preliminaries

In this section some concepts about road network and propositions of path search regions on a road network are recapitulated.

3.1 Road network, route and computation point

A road network with nodes and links representing the cross-points and road segments can be regarded as a graph

$$G : G = (V, L),$$

where *V* is a set of vertices $\{ v_1, v_2, ...v_n \}$, and *L* is a collection of edges $\{ l_1, l_2, ...l_m \}$, used to indicate the relationship between ver-

tices. For example, if the vertices v_p and v_q are related, the relation would be indicated by an edge that will be designated as

$$l_i = (v_p, v_q).$$

The predefined route from a start point v_s to an end point v_e is given by an array

$$Route(v_s, v_e) = \{(v_{r1}, ..., v_{ri}, ..., v_{rn}) | v_{r1} = v_s, v_{rn} = v_e,$$

$$v_{ri} \in V, l_j = (v_{ri-1}, v_{ri}), l_j \in L, i = 2, ..., n-1\}.$$

A sub-route of $Route(v_s, v_e)$ is defined as $(v_{rl}, ..., v_{rj})$, which overlaps with $Route(v_s, v_e)$. If the target object set is $T = \{t_a, t_b, ...\}$ and $t_i \in T$ with a corresponding node $v_{ti} \in V$, the NN for v_{ri} on $Route(v_s, v_e)$ is t_i when the shortest path

$$Path_{Vri\_ti} = \{min(v_{ri}, ..., v_j, ...v_{ti}) | v_j \in V, ti = t_a, t_b, ...\}.$$

[Definition 1] *Node $v_d \in V$ is called the divergence point between $Route(v_s, v_e)$ and $Path_{Vri\_ti}$ if:*

1) *The sub-route $(v_{ri}, ..., v_d)$ of $Path_{Vri\_ti}$ is also a sub-route of $Route(v_s, v_e)$;*

2) *The node following v_d along $Path_{Vri\_ti}$ is not on $Route(v_s, v_e)$.* □

Consider Figure 1: t_e is NN of node S on $Route(S, E)$, the shortest path from S to t_e is P_1. The divergence point between $Route(S, E)$ and P_1 is Q_1, and is just the point in which the shortest path branches off the route. It is obvious that the points on the sub-route $(v_{ri}, ..., v_d)$ share the same NN t_i. Therefore, there is no need to search NN for every point on the sub-route; CNN search can be regarded as a series of NN searches for some points on the route; and those points are called computation points.

[Definition 2] *Node $v_c \in V$ is called the computation point of $Route(v_s, v_e)$ if:*

1) *v_c is the start point of $Route(v_s, v_e)$;*

2) *v_c is on $Route(v_s, v_e)$ and is also a node on the route following a divergence point between $Route(v_s, v_e)$ and $Path_{Vri\_ti}$.* □

In Figure 1: S is a computation point; and C_1 is a computation point following the divergence point Q_1. NN for C_1 is t_d. t_d is also regarded as the NN for all the points on the sub-route of $Route(v_s, v_e)$ from the previous divergence point Q_1 (except Q_1 itself, its NN is t_e) to the following divergence point Q_2. This is because on the real road network we can branch off the route only on cross-point.

3.2 Path search regions

To solve the CNN problem, there are two main issues: one is the selection of computation point on the route; and another is the computation of NN for the computation point. Here, we give two propositions on the road network for nearest object search.

[Proposition 1] *For a source point S and a target object t, when the length of a path from S to t is r, any target object which is nearer to S than t can only be found inside a circle region, denoted as r-region, whose center is S and whose radius is r (Figure 2(a)).* □

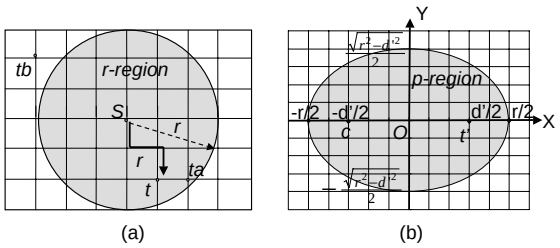

Figure 2 (a) Nearest target search region: *r-region*; (b) Shortest path search region: *p-region*.

We leave the proof out in this paper, as it can be convinced by the fact that any road segment outside *r-region* can only lead to a path longer than r from S.

[Proposition 2] *For two points S and t on the road network with straight-line distance d, the test of whether there is a path shorter than r from S to t can be based on a path search region, denoted as p-region, the sum of the straight-line distance between any nodes inside this region and S and that between this node and t is not longer than r.* □

For an easy description we define a coordinate for them in Figure 2(b):

$$p - region =$$

$$\{(x, y) | \sqrt{(x + d/2)^2 + y^2} + \sqrt{(x - d/2)^2 + y^2} \leqq r\}.$$

The origin *O* of the coordinate is on the center of line $\overline{St}$, the x-axis passes along line $\overline{St}$, and the y-axis is perpendicular to the x-axis on the origin *O*. To find the shortest path from *S* to *t* is based on the road segments inside the region (as the grey ellipse in Figure 2(b)). This means that if there is any path shorter than *r* from *S* to t all the road segments on this path could only be found inside *p-region*. The region can also be simplified to a rectangle with length *r* and width $\sqrt{r^2 - d^2}$.

4. *k*-CNN Search Method

The problem of *k*-CNN search which we address in this paper is to find *k*-NN's for any point along a specific route on a large road network. *k*-NN's are the first *k* target objects from the point on the route on the sort of the shortest path length.

4.1 Bounds of the shortest path length

Observe Figure 3: in 2-CNN search, 2-NN's for the first computation point *S* are *t* and t_g. To find 2-NN for the next computation point *c* can take advantage of the previous computation, for example at least *t* and t_g can be regarded as 2-NN up to now, which are with the possible longest paths from *c*: $(Path_{cq} + Path_{qt})$ and $(Path_{cq} + Path_{qt_g})$. However, the real path length from *c* to them may be varied. This is because there may be some shorter paths,

and the lower and upper bounds of the path length from c to t are r_{min} and r_{max}, which can be decided by

$$r_{min} = |Path_{cq} - Path_{qt}|. \quad (1)$$

$$r_{max} = Path_{cq} + Path_{qt}. \quad (2)$$

and those for t_g are:

$$r_{ming} = |Path_{cq} - Path_{qtg}|. \quad (3)$$

$$r_{maxg} = Path_{cq} + Path_{qtg}. \quad (4)$$

It means that t is a NN candidate for c with a possible path length varied from r_{min} to r_{max}. The value of $Path_{qt}$ has been computed in the previous NN search step for S, and the value of $Path_{cq}$ is the curve length between *c* and *q*. Though r_{maxg} is greater than r_{max}, it can to say the path length to t_g is long than that to *t*.

Based on **Proposition 1**, if *r-region* is decided for C with the radius r_{maxg}, the target objects nearer than the up-to-now known 2-NN's can be found only inside this *r-region*.

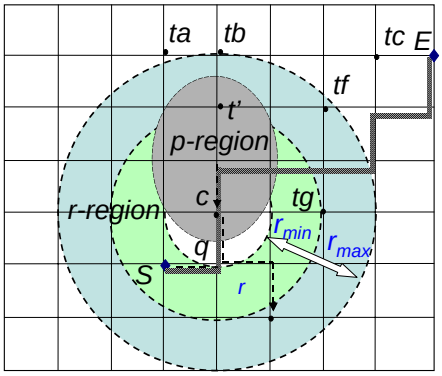

Figure 3 Search regions generated for a new computation point based on NN of the previous computation point.

4.2 Data structures for *k*-CNN search

The NN search process bases on the R-tree index and a priority queue *Queue* [12]. *Queue* is used to record the intermediate results: the candidate targets or the internal nodes of R-tree inside *r-region*. The key used to order the elements on *Queue* is the straight-line distance of R-tree node and the path length computed for target object. *Queue* is initialized as a node of the R-tree, which overlaps with *r-region*. When a target object turns on the head of the priority queue, it becomes the candidate for further computation: the path length is computed for the candidate based on the search region. If the path length is smaller than the key of the head element of the queue, the candidate is the result. Otherwise, when the path length is smaller than the radius of search region, the search region is reset with the new length as the radius. The value of radius is decreased by keeping step with the ongoing path length computation for the candidates, and the search region is adjusted until there is no candidate inside it.

In the process of searching *k*-NN's for a computation point, the priority queue can also be used. When an object with the computed path length turns out on the head of the queue, the first (or the nearest) neighbor is found. On the next time, the second (2-nearest) neighbor will be returned. However, in the process of CNN search, the priority queues for the computation points except the start point on the route only record the nodes or objects with the shorter path than *r*. In other words, it assures the objects found on the head of queue are in order, but cannot assure that there are enough (here, *k*) objects in the queue. To solve this problem, another data structure, called *k*-queue, is adopted to keep the up-to-now *k* candidates and distances (or distance bounds) for *k*-NN search. The greatest upper bound of distance is regarded as a cutoff value for pruning nodes or objects: a node or an object with a longer path is not inserted into the priority queue, while there are at least *k* candidates kept in the priority queue. *k*-queue is defined as a fixed length queue, where

- *k* is the number of nearest neighbors found for every point on the predefined route. It is decided at the beginning of *k*-CNN search, and be kept in the process of the search.
- The elements inside the queue are triplets $< t, r_{min}, r_{max} >$, where t is a target object and r_{min} and r_{max} are the lower and upper bounds of the path length from the current computation point to t. If the real path length to t has been computed, then r_{min} and r_{max} are set as the same value. There are at most *k* elements in the queue.
- The longest r_{max} in *k*-queue is regarded as a cutoff value, which is used to set *r-region* for pruning objects in the priority queue *Queue* .

For the computation points on the route except the start point, the contents of *k*-queue are initialized as: the *k*-NN's of the previous computation point. The lower and upper bounds of the path length for every object recorded in the queue are r_{min} and r_{max}.

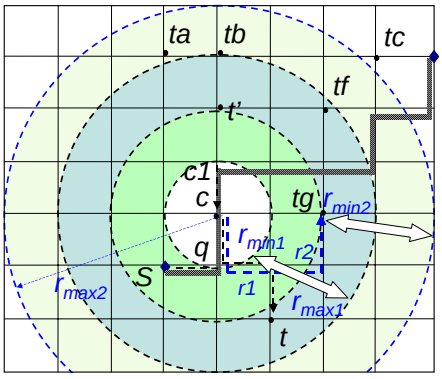

Figure 4 There are lower and upper bounds of path length from the current computation point to the previous k-NN's.

For a 2-CNN search given in Figure 4, the 2-NNs for S are t and t_g, then the *k*-queue for c is like $< t, 1, 3 >, < t_g, 2, 4 >$. Which means 2-NNs of *c* can only be found inside a *r-region* whose center is *c* and radius is 4 (the upper bound of t_g). Therefore, the *k*-NNs for c can be found by testing the objects in the priority queue $Queue$

and those inside *k*-queue.

Considering that, the distance between internal nodes of R-tree is shorter than that between two objects inside the two nodes, and the distance between the two objects is shorter than the path length between the two objects on road network. The elements in *k*-queue are on the order of path length, *r-region* and *p-region* are determined by the path length, and the elements in the priority queue are sorted on the three kinds of length. The loose condition for the test of tree nodes becomes stricter and stricter for testing of distance between objects and path length from one object to another.

4.3 About *p-region*

Based on **Proposition 2**, to test whether the path length from c to a candidate t' inside *r-region* is shorter than r_{max} can be based on a *p-region*: the path length from the current computation point *c* to NN of the previous computation point is r_{max}, and the straight-line distance between *c* and the candidate node t' is d'. *p-region* for the computation of the shortest path from c to t' is the white ellipse in Figure 3.

5. Algorithm for *k*-CNN search on road network

The problem of *k*-CNN search along a specific route on road network is to find out *k*-NN's for any place on the route in order.

In the algorithm of NN-search, a *Queue* is used to record the nodes in R-tree whose child nodes or referred objects are visited in order of distance from the computation point. The *Queue* ensures an incremental generation of NN's in distance order for one computation point. However, to compute the *k*-NN's for the next computation point cannot be based on the NN of the current computation point. So a new queue $Queue_k$ is adopted to record the first *k*-NN's of a computation point. The *k*-queue, $Queue_k$, is a queue with fixed length of *k*. The algorithms for *k*-CNN search are given in the following:

==========

Algorithm k**-CNN-search**

/* Input: route [S, E], target object set T

Output: Result set of triples

$\{< (nn_1, ..., nn_k), interval, (path_1, ..., path_k) >$,...$\}$*/

1. Initialize:

set first computation point: CP = S;

set NN search region for CP: r = Max;

initialize $Queue_k$ as a queue with fixed length *k*;

2. Do steps 3 to 7 until CP equals to E;

3. Call *k*-NN-search with CP, r and $Queue_k$; and get a triple of

$< (nn_1, ..., nn_k), [CP, q], (Path_{CPnn1}, ..., Path_{CPnnk}) >$;

4. Replace interval [CP, q] with [q_{pre}, q],

insert $< (nn_1, ..., nn_k), [q_{pre}, q], (Path_{CPnn1}, ..., Path_{CPnnk}) >$

into Result set;

5. Generate next computation point CP:

CP = next intersection following q along route;

6. For i = 1 to k do

$r_{maxi} = Path_{CPq} + Path_{qnni}$;

$r_{mini} = |Path_{CPq} - Path_{qnni}|$;

insert (nn_i, r_{mini}, r_{maxi}) into $Queue_k$;

7. Set *k*-NN search region for CP:

r = Max(r_{maxi});

==========

The algorithm of *k*-CNN search calls *k*-NN search procedure to find *k* nearest neighbors for the route. It starts from searching *k*-NN's for the start point of the route (step 1, and step 3), decides the next computation points (step 5) based on the previous results, adopts $Queue_k$ to keep the up-to-now *k*-NN's for the current computation point (step 6),and last generates the search region parameters (step 7) for new computation point. The loop from steps 3 to 7 stops after the search of *k*-NN for the end point of the route.

==========

Algorithm k**-NN-search**

/* Input: route [S, E], target object set T;

source point CP, search region r and $Queue_k$;

Output: a triple

$< (nn_1, ..., nn_k), interval, (path_1, ..., path_k) >$*/

1. Initialize priority queue *Queue* as the root of R-tree and result-number as 0;

2. Locate first node overlapping with region r

on R-tree, compute straight-line distance d between

it and CP, and insert node into *Queue*;

3. Do steps 4 to 5 unless *Queue* is Null or result-number is *k*;

4. If head of *Queue* is leaf node of R-tree,

Then for the object which with a shorter d in *Queue*

or a shorter r_{mini} in $Queue_k$, do

(1) initialize *p-region* with Max(r_{maxi}) for the object selected from *Queue*;

initialize *p-region* with its r_{max} for the object selected from $Queue_k$;

(2) Call SP-search to compute shortest path SP

from CP to it,

(3) insert result to *Queue* and $Queue_k$;

Else

(1) compute straight-line distance between them,

(2) insert result to *Queue*;

5. If head of *Queue* is leaf node with computed

shortest path

/*object t of leaf node is NN for CP, computed path

$Path_{CPt}$ is shortest path from CP to t*/

Find divergence q of $Path_{CPt}$ and route;

Insert triplets of $(t, length(Path_{CPt}), length(Path_{CPt}))$ into $Queue_k$;

result-number ++;

Reset r as length($Path_{CPt}$) of the tail element of $Queue_k$;

6. Return result of *k* triples $< t, [CP, q], Path_{CPt} >$

where t is inside $Queue_k$;

==========

The algorithm of *k*-NN search is an extension of NN-search al-

gorithm used in CNN search [8]. The *Priority queue* plays the same role as that in CNN search. The *k*-queue ($Queue_k$) is used to keep the up-to-now nearest neighbors for the current computation point and the path length kept in the tail element of $Queue_k$ is the cutoff value of the following search. When the computation point is not the start point of the route, a new object is inserted into $Queue_k$ on the order of the path length and the cutoff value is also updated after every insertion (step 5). And so *r-region* and *p-region* are reduced with the update: the cutoff value is set as the radius of *r-region* in the following test.

The k-NN search stage ends when one of the following conditions is satisfied: 1) *Queue* becomes empty (step 5), or 2)*k* or more results have been returned (step 5). The *cutoff value* is initialized as the greatest upper bound of objects in $Queue_k$ and adaptively corrected during the algorithm processing: in step 4 (3), by inserting a new tested object into $Queue_k$, the *cutoff value* may be also updated. Because there are only *k* elements in $Queue_k$, any candidate tested inside *r-region* leads to a longer path than the current farthest object will not be inserted into $Queue_k$. Furthermore, any object with a shorter path length than the cutoff value is inserted into $Queue_k$, the element on the tail of $Queue_k$ is dropped and the cutoff value is reset as the new one. The search can be done in a stepwise optimization way.

6. Conclusion

From a viewpoint of decreasing the times of disk access, we have proposed a method for CNN search on the large hierarchical road network by minimizing the search region. However, that method cannot support a *k*-CNN search, which finds more nearest neighbors than one. We adopt a fixed length queue for recording up-to-now intermediate results to solve this problem. Because the search regions can be reduced stepwise based on the intermediate results, the *k*-CNN search can be realized efficiently. Our method is based on the premise of that the distance from one place to another on the road network is the path length of them. And so the search regions can be decided based on the path length, then the filtering condition in the searching via hierarchical data structures can take advantage of it. In ITS applications, CNN, *k*-NN or *k*-CNN search is usually based on the travel cost, sometimes dynamical values, from one place to another, and the travel cost may not be in direct proportion to their path length or the straight-line distance. This problem cannot be solved by the method proposed in this paper.

Acknowledgments

Our research is partly supported by the 21st Century COE (Center of Excellence) Program for 2002, a project titled Intelligent Media (Speech and Images) Integration for Social Information Infrastructure.

References

[1] G.R. Hjaltson and H. Samet: "Incremental Distance Join Algorithms for Spatial Databases", *Proc.of ACM-SIGMOD*, pp. 237–248 (1998).

[2] J. A. Orenstein: "A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces", *Proc. of ACM-SIGMOD*, pp. 343–352 (1990).

[3] H. Shin, B. Moon and S. Lee: "Adaptive and Incremental Processing for Distance Join Queries", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 15, No. 6, pp. 1561–1578 (2003).

[4] J. Feng and T. Watanabe: "Search of Continuous Nearest Target Objects along Route on Large Hierarchical Road Network", *Proc. of Data Engineering Workshop* (2003).

[5] Y. F. Tao, D. Papadias and Q. M. Shen: "Continuous Nearest Neighbor Search", *Proc. of VLDB'02*, pp. 287–298 (2002).

[6] Z. X. Song and N. Roussopoulos: "K-Nearest Neighbor Search for Moving Query Point", *Proc. of SSTD'01*, pp. 79–96 (2001).

[7] S. Bespamyatnikh and J. Snoeyink: "Queries with Segments in Voronoi Diagrams", *SODA* (1999).

[8] J. Feng and T. Watanabe: "A Fast Method for Continuous Nearest Target Objects Query on Road Network", *Proc. of VSMM'02*, pp. 182–191 (2002).

[9] E. H. Jacox and H. Samet: "Iterative spatial join", *ACM Trans. Database Syst.*, Vol. 28, No. 3, pp. 230–256 (2003).

[10] D. H. Lee and H. J. Kim: "An Efficient Technique for Nearest-Neighbor Query Processing on the SPY-TEC", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 15, No. 6, pp. 1472–1486 (2003).

[11] G.R. Hjaltson and H. Samet: "Distance Browsing in Spatial Databases", *ACM Transactions on Database Systems*, Vol. 24, No. 2, pp. 265–318 (1999).

[12] J. Feng and T. Watanabe: "A Fast Search Method of Nearest Target Object in Road Networks", *Journal of the ISCIE*, Vol. 16, No. 9, pp. 484–491 (2003).