Kim HI, Chang JW. k-nearest neighbor query processing algorithms for a query region in road networks. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 28(4): 585–596 July 2013. DOI 10.1007/s11390-013-1359-8

k-Nearest Neighbor Query Processing Algorithms for a Query Region in Road Networks

Hyeong-Il Kim[1] and Jae-Woo Chang[1,2,*]

[1] *Department of Computer Engineering, Chonbuk National University, Chonju 570-752, Korea*

[2] *Cloud Open R&D Center, Chonbuk National University, Chonju 570-752, Korea*

E-mail: {melipion, jwchang}@jbnu.ac.kr

Received September 9, 2012; revised May 6, 2013.

Abstract Recent development of wireless communication technologies and the popularity of smart phones are making location-based services (LBS) popular. However, requesting queries to LBS servers with users' exact locations may threat the privacy of users. Therefore, there have been many researches on generating a cloaked query region for user privacy protection. Consequently, an efficient query processing algorithm for a query region is required. So, in this paper, we propose k-nearest neighbor query (k-NN) processing algorithms for a query region in road networks. To efficiently retrieve k-NN points of interest (POIs), we make use of the Island index. We also propose a method that generates an adaptive Island index to improve the query processing performance and storage usage. Finally, we show by our performance analysis that our k-NN query processing algorithms outperform the existing k-Range Nearest Neighbor (kRNN) algorithm in terms of network expansion cost and query processing time.

Keywords island index, k-nearest neighbor query processing scheme, location-based service, road network

1 Introduction

Recently, many applications based on users' location information are rapidly diffused with the development of wireless communication technologies and the popularity of smart phones. Such applications are named LBS (location-based services) since they provide additional services based on the users' exact locations obtained by location positioning devices, e.g., GPS. LBS applications include buddy search, location-based tourist information, route guidance, and retrieval of nearest POIs (points of interest) like restaurants and gas stations, etc. However, a user's private information is in danger because the exact location of the user should be sent to an LBS server to enjoy such applications. If an adversary or malicious LBS provider abuses this information, it can speculate the user's private information like lifestyle, disease and religion by knowing when/where the user frequently visits. Actually, some cases of the stalking or the leakage of personal information have occurred[1-2] by using users' location information of LBS. Therefore, a mechanism for users' privacy protection is required for the safe use of LBS.

To protect a user's location privacy, cloaking area creation schemes[3-6] have been actively studied. They guarantee the privacy protection by generating and sending a cloaking area (or a query region) satisfying k-anonymity property to the LBS server, instead of using the user's exact location. The k-anonymity property is simply defined as a mechanism where a query region includes not only a user who requests a query, but also the $k-1$ other users nearby him/her. By using the cloaking area creation schemes, the location exposure probability of a query issuer can be reduced by $1/k$. This is because an adversary or a malicious LBS provider cannot distinguish the query issuer among other users inside the query region. However, in this case, as the LBS provider cannot know the exact location of a query issuer, it should perform query processing under a premise that a user can be anywhere inside the query region. In other words, the LBS provider should process the query over a query region, not an exact query location.

Therefore, we, in this paper, consider a k-nearest neighbor (k-NN) query which returns k nearest POIs from the location of a query issuer in road networks. For example, a user can issue a query like "send me the three nearest restaurants from my current location"

Regular Paper
This research was supported by the Korea Institute of Science and Technology Information (KISTI).
The preliminary version of the paper was published in the Proceedings of EDB2012.
*Corresponding Author

with a query region (not an exact point). Then, an LBS provider processes the query and returns a set of restaurants that can be candidates for the any points inside the query region. By doing this, the user's location privacy can be protected. There are two existing schemes[7-8] that process k-NN query over a query region. However, both schemes suffer from the following problems. First, they should be extended to road segments where POIs are actually located, thus leading to the degradation of POI retrieval performance due to a lot of network expansion. Secondly, they need to carry out an additional calculation to obtain the distances from a node to POIs being located on the adjacent road segment of the node.

To solve the problems, we propose IB_kRNN (Island-Based k-Range Nearest Neighbor) query processing algorithm and an adaptive IB_kRNN (AIB_kRNN) query processing algorithm. To support fast query processing, we first propose IB_kRNN by expanding the Island[9] scheme which pre-computes the distances between nodes and nearby POIs. In the Island scheme, every node has an Island index which stores pre-computed distances to POIs being located inside a given radius from the node. By adapting the Island index, IB_kRNN rapidly processes a k-NN query over a query region. Next, we propose AIB_kRNN to improve IB_kRNN. To increase the efficiency of Island index, we generate an adaptive Island index by applying different radiuses to each node based on the POI density of each node.

The contributions of this paper are as follows. First, we design two query processing algorithms (IB_kRNN and AIB_kRNN) for a query region in road networks. Because both algorithms make use of the Island index that can be loaded on the main memory, they can rapidly process a k-NN query. Second, we devise a method to adaptively generate the Island index in AIB_kRNN algorithm. By considering the POI density of each node, we can manage a trade-off between the size of the index and the number of POIs. Finally, we provide experimental results showing that our schemes outperform an existing scheme k-Range Nearest Neighbor (kRNN)[8] in terms of the number of POIs retrieved and query processing time.

The rest of the paper is organized as follows. Section 2 introduces related work and Section 3 describes the motivation and the problem setting of our work. The proposed k-NN query processing algorithms over a query region in road networks are presented in Sections 4 and 5. The extensive experimental evaluation of our algorithms is presented in Section 6. Finally, Section 7 concludes the paper with further research.

2 Related Work

We first review cloaking area creation schemes both in Euclidean space and in road networks, and then introduce existing k-NN query processing algorithms over a query region in road networks.

2.1 Cloaking Area Creation Schemes

Cloaking schemes which generate a cloaking area satisfying k-anonymity property in Euclidean space have been actively studied[3-6]. First, Mokbel *et al.*[3] proposed the New Casper scheme which creates a cloaking area by using a grid-based hierarchical pyramid data structure. Thus, New Casper scheme generates cloaking areas with different sizes, according to the density of mobile users. Secondly, Ghinita *et al.*[4] proposed the MobiHide scheme which transforms the locations of mobile users into 1-dimensional coordinates by using a Hilbert curve. Then, MobiHide makes up a distributed hash table called Chord and creates a cloaking area. Thirdly, Kim *et al.*[5] proposed a distributed scheme where a query issuer generates a cloaking area in a distributed manner by collaborating with other mobile users. Finally, Lee *et al.*[6] proposed the GCC scheme which creates a cloaking area by calculating the privacy protection level of a cloaking area by using entropy. Thus, it can reduce the location exposure probability of a query issuer. However, because cloaking area creation schemes in Euclidean space cannot reflect the road network, inappropriate query results may be returned to a query issuer. The reason is that the returned result is apart from the location of a query issuer in case a network distance is quite far from the location of the query issuer.

To the best of our knowledge, the XStar scheme proposed by Wang *et al.*[10] is the only work which creates a cloaking area in road networks. XStar considers not only k-anonymity property, but also l-diversity property to guarantee the privacy protection of mobile users. l-diversity property is defined as a mechanism where a cloaking area includes not only road segment that a query issuer is located on, but also the $l-1$ other adjacent or nearby road segments. By doing this, the exposure probability of road segments can be reduced by $1/l$. In other words, because a cloaking area created by the XStar scheme includes not only more than k users but also more than l road segments, XStar can hide the identity of a query issuer and a road segment where the query issuer is located. The XStar scheme first allocates a query issuer into a nearby star node being connected with more than three road segments. Then it checks whether the number of users allocated to the star node satisfies the required k-anonymity or not. Next it examines whether the number of road segments crossing the selected star node satisfies the required l-diversity. If both privacy requirements are satisfied, XStar crea-

tes a star node with its adjacent road segments as a cloaking area.

2.2 *k*-NN Query Processing Algorithms over Query Region

When a user requests a k-NN query by using a cloaking area, an LBS provider should perform query processing under a premise that the user can be anywhere inside the query region. This is because the LBS provider cannot know the exact location of a query issuer inside the cloaking area. Some studies[11-14] on a k-NN query processing algorithm for a query region have been performed. First, Kalnis *et al.*[11] proposed the CkNN (Circular Range k-NN) algorithm which considers a circular query region. CkNN scheme uses the R-tree to find POIs over the circular query region. Secondly, Chow *et al.*[12] proposed the ARNN (Approximate Range Nearest Neighbor) algorithm. ARNN finds the nearest neighbor for a given cloaking region by using a Voronoi diagram. Thirdly, Xu *et al.*[13] proposed the kCRNN (Circular-Region-Based kNN) algorithm. kCRNN applies a filtering method based on distance measure to efficiently prune out POIs. Finally, Um *et al.*[14] proposed a query processing algorithm which makes use of a Voronoi diagram for retrieving the nearest POIs efficiently. This algorithm adopts 2D-coodinate scheme to prune POIs. However, these schemes operate on the Euclidean space. Thus, they are unable to provide the high quality of service to the query issuer because they cannot consider the road networks where the query issuer actually moves along. For example, assume that a user requests a query on the left side of the river. Then, POIs located on the right side of the river can hardly be query results unless there exists a bridge nearby the user. Therefore, it is necessary to process queries based on road networks to provide good services to LBS users.

There exist two algorithms considering road networks. First, Ku *et al.*[7] proposed the PSNN (Privacy Protected Spatial Network Nearest Neighbor Query) algorithm. PSNN retrieves all POIs inside the given cloaking area and finds POIs outside the cloaking area through road network expansion. Secondly, Bao *et al.*[8] proposed the kRNN (k-Range Nearest Neighbor) algorithm. kRNN pre-computes the distances of every pair of nodes to reduce the cost of network distance calculation. kRNN retrieves all POIs inside the given query region and computes distances which are shared by every pair of expansion border points where an expansion border point is a boundary node of the cloaking area. Then, kRNN repeatedly expands a network from one of the expansion border points.

3 Problem Setting

In this section, we present the motivations of our research, describe the system architecture for LBS, and define notations used in this paper.

3.1 Motivations

The existing k-NN query processing algorithms over a query region in road networks have some drawbacks. PSNN[7] suffers from the duplicate expansion of a road network, thus causing the degradation of the query processing performance. kRNN[8] solves the problem of duplicate expansion and improves query processing performance by pre-computing the distances between nodes of a road network. But it still needs to be expanded to deal with road segments where POIs are actually located. kRNN also remains to be extended to carry out an additional calculation to get distances from a node to nearby POIs in the adjacent road segments of the node. The problems cause the degradation of the retrieval performance of kRNN.

Therefore, in this paper, we propose both IB_kRNN (Island-Based k-Range Nearest Neighbor) query processing algorithm and an adaptive IB_kRNN (AIB_kRNN) query processing algorithm to tackle these problems. First, IB_kRNN rapidly processes k-NN query over a query region because it adopts an Island technique[9] which pre-computes the distances between a node and nearby POIs. Secondly, AIB_kRNN utilizes an adaptive Island index which is generated by considering the POI density of each node. In other word, AIB_kRNN adaptively applies different radiuses to each node. By doing this, the proposed schemes can reduce the number of average network expansion. Because the Island index can be stored in the main memory in both schemes, our schemes guarantee fast query response time for a query issuer.

3.2 System Architecture

There exist two types of system architectures in LBS: centralized and distributed. The main difference between both architectures is who generates the cloaking area. In a centralized system, an anonymizer generates a cloaking area, whereas a query issuer generates a cloaking area by communicating with other mobile users in a distributed system. Our query processing schemes can be applied to both system architectures. However, in this paper, we only consider the centralized system architecture for the sake of a simple explanation.

Fig.1 shows a general centralized architecture for LBS. The system mainly consists of mobile clients, an anonymizer, and an LBS provider. Mobile clients are users who request LBS queries. To enjoy a location-

based service, mobile users send to the anonymizer a query with their exact location, the number of POIs that users want to receive, k-anonymity, and l-diversity. The anonymizer is a trusted third party who generates a cloaking area, i.e., a query region, by considering k-anonymity and l-diversity. Here, k-anonymity and l-diversity means the number of users and the number of road segments that the query issuer wants the cloaking area to include, respectively. Then, the anonymizer sends the cloaking area to the LBS provider. An LBS server plays an important role in processing the query over the query region to retrieve POIs. At this time, the LBS provider returns the candidates including an actual result back to the anonymizer because the LBS provider does not know the exact location of the user in the cloaking area. Finally, the anonymizer returns to the query issuer the final result based on the exact location of the query issuer. Among this procedure, we focus on the LBS provider who performs query processing over a query region in road networks. Therefore, we design query processing algorithms to efficiently respond to the query issuer.

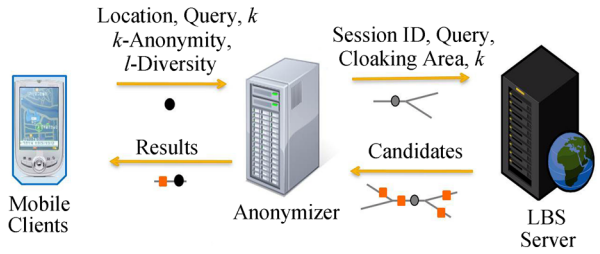


Fig.1. General centralized system architecture for LBS.

3.3 Notations

We model the underlying road network as an undirected weighted graph $G = (V, E)$ where V is a set of vertices and E is a set of edges which mean road segments. A query is defined as $Q = (R, k)$ where R is a given query region and k is the number of POIs that the query issuer wants to receive. A query region R is defined as $R = (\text{seg}_1, \text{seg}_2, \ldots, \text{seg}_n)$ where seg_i $(1 \leqslant i \leqslant n)$ are road segments that compose the query region. In this paper, query processing is normally performed over a query region R by considering a road network. We manage three types of indices: node index, road segment index, and Island index. These indices are stored into the main memory so that the proposed algorithms can rapidly process a query. First, we define the node index as shown in Definition 1. The node index is used to support fast query processing when the network expansion is performed for adjacent nodes.

Definition 1. *Node index is defined as* $V = (ID, x, y, adj\_node, adj\_edge, degree)$ *where ID is an identity of a node, x and y are the coordinates of the node, adj_node and adj_edge are the list of adjacent nodes and road segments, respectively, and degree is the number of road segments being adjacent to the node.*

Secondly, we define the road segment index as shown in Definition 2. This index is helpful to rapidly find POIs when retrieving road segments inside a query region.

Definition 2. *Road segment index is defined as* $E = (ID, node_S, node_E, dist, poicount, POI)$ *where ID is an identity of a road segment,* $node_S$ *and* $node_E$ *are start and end nodes being connected by a road segment, dist is the length of a road segment, and poicount and POI are the number of POIs being located on a road segment and a list of POIs, respectively.*

Finally, we define Island index as shown in Definition 3. We generate an Island index for each node by finding POIs within a given range from a node and precomputing the distances from the node to found POIs. Here, POIs are stored with their network distances from the node in increasing order. By accessing the Island index, we do not need to expand a road segment that POIs are actually located on. Therefore, the number of network expansions can be reduced.

Definition 3. *Island index is defined as* $I = ((P_1, dist_1), (P_2, dist_2), \ldots, (P_p, dist_p))$ *where* P_i $(1 \leqslant i \leqslant p)$ *is the identity of a POI being within a given range from a node, and* $dist_i$ *is the corresponding network distance from the node to each POI.*

We assume that a query region is a set of road segments and there exists one or more nodes that are connected with a road segment outside the query region. From the nodes, we can expand a road network to retrieve POIs. We define the nodes as expansion border points in Definition 4.

Definition 4. *Expansion border point is a node which is connected with a road segment outside a query region. Each expansion border point* (B_v) *manages DistToBorder and BorderResult where DistToBorder is a table that stores the network distances to other expansion border points and BorderResult is a table that stores a current k-NN query result for the expansion border point.*

To expand a road network, we need to find out nodes that exist outside a query region but are adjacent to the query region. We define the nodes as expansion candidates in Definition 5.

Definition 5. *Expansion candidates are nodes that are outside a query region while are adjacent to the query region. Expansion candidates are arranged in the increasing order of network distances from the query region. Among all expansion candidates, the node with the shortest distance to the query region is termed* V_{selected}.

4 Island Index Based Query Processing Algorithm

In this section, we present our IB_kRNN (Island-Based k-Range Nearest Neighbor) algorithm that deals with a k-NN query for a query region in road networks. First, IB_kRNN generates an Island index for all nodes in the road network. Then, IB_kRNN retrieves all POIs inside a query region and sets expansion border points. Secondly, it searches POIs outside the query region by accessing the Island index. Thirdly, it finds other POIs through network expansion from expansion border points, in order to guarantee an accurate query result. Finally, it returns the final result to a query issuer.

4.1 Step 1: POI Retrieval Inside a Query Region

An LBS provider cannot know the exact location of a query issuer inside a query region being sent from anonymizer. This is why the LBS provider has to process the query under a premise that the query can be anywhere inside the query region. This means that all POIs inside the query region should be returned to the query issuer. Therefore, in this step, we first retrieve all POIs located on road segments that form the query region. This can be easily done by utilizing a road segment index. Then, we determine a set of expansion border points, $B = (B_{V1}, B_{V2}, \ldots, B_{Vm})$ where m is the total number of expansion border points for the given query region. Next, we calculate the distances among all expansion border points and store them into their DistToBorder tables. Finally, we store into BorderResult the current k-NN query results of all expansion border points with their network distances.

Fig.2 gives an example of POI retrieval inside a query region. Assume that a query region is given as a set of road segments which are depicted as thick lines, i.e., seg_{AB}, seg_{BC}, seg_{BD}, and seg_{BE}. At first, our IB_kRNN algorithm searches POIs located on the road segments within the query region. Therefore, our algorithm finds P_1 on seg_{BE} and P_2 on seg_{BD} by accessing the road segment index from these road segments. Then, it regards nodes A, C, and E as expansion border points (B_A, B_C, and B_E). Note that nodes B and D are not expansion border points because they are not connected with other road segments outside the query region. Next, it calculates the distances among all expansion border points and stores them into their DistToBorder as shown in Table 1. In case of B_A, the network distance with B_C is 8 whereas the distance with B_E is 10. Finally, all expansion border points maintain their current k-NN query results with their network distances into BorderResult in Table 1. As two POIs, i.e., P_1 and P_2, are found, every expansion border point stores them as the current 2-NN results.

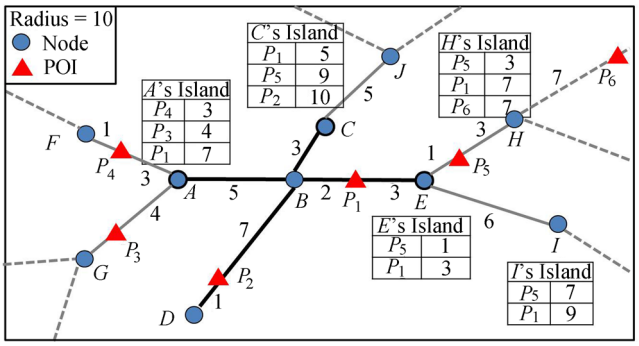


Fig.2. Example of POI retrieval inside a query region.

Table 1. Expansion Border Points Information

Expansion Border Points	DistToBorder			BorderResult		
	B_A	B_C	B_E	k_1	k_2	k_3
B_A	0	8	10	$(P_1, 7)$	$(P_2, 12)$	∞
B_C	8	0	8	$(P_1, 5)$	$(P_2, 10)$	∞
B_E	10	8	0	$(P_1, 3)$	$(P_2, 12)$	∞

4.2 Step 2: POI Retrieval from Island Index

Once a set of expansion border points, $B = (B_A, B_C, B_E)$, are found in step 1, our algorithm retrieves other POIs by searching the Island index from expansion border points. This step can be categorized into two cases according to whether or not an Island index contains less POIs than k that a user requests. When an Island index stores the same or more POIs than k, our algorithm can obtain k closest POIs by accessing only the Island index. Therefore, it is very important to determine how many POIs should be stored in the Island index so that we can perform k-NN query processing in an optimal way. For this, it is possible to obtain the near optimal value as the number of POIs to be stored in the Island index of a node. This can be done by using the sampled POIs which are large in size enough to calculate the optimal value and are generated from a real road map (e.g., San Francisco map). The detailed explanation is omitted here due to the space restriction. Next, our algorithm updates the BorderResult of all expansion border points and terminates the query processing at the node. This is because the Island index of a node stores pre-computed distances to POIs being located inside a given radius from the node in an increasing order. To update distances stored in the BorderResult of other expansion border points with the found POIs, it is necessary to gain accesses to DistToBorder which contains the distances between other expansion border points and the distances from the found

POIs to the node, i.e., dist_p. If the summation of the two values is smaller than the current result of each expansion border point, our algorithm updates the BorderResult with this value. Once the query processing at a node is finished, no more updates occur at this node because the result for the node is guaranteed by the pre-processed Island index.

For example, we start from an expansion border point B_A. Assume that a radius for generating the Island index is 10, a user requests three POIs ($k = 3$), and the Island indices are shown in Fig.2. Because the Island index of B_A stores three POIs which equals the k value requested, our algorithm finds three closest POIs, i.e., P_4, P_3, and P_1. Among them, P_4 and P_3 with shorter distance than the current result of B_A are inserted into the BorderResult of B_A. Next, it updates the BorderResult of all expansion border points and terminates the query processing at B_A. The update at each expansion border point is done as follows. The distance from B_C to P_4 is calculated by summing 8 in DistToBorder of Tables 1 and 3 in the Island index of B_A. Because the calculated distance 11 is smaller than the current third closest result of B_C (i.e., ∞), P_4 is inserted into the BorderResult of B_C. Meanwhile, the distances from B_C to P_3 and P_1 are 12 and 15, respectively. Because the distances are larger than the current value of B_C, they are not reflected into the BorderResult of B_C. Similarly, P_4 is inserted into the BorderResult of B_E with the distance 13 ($10+3$). Table 2 shows the result after retrieving the Island index of B_A. The newly updated values are highlighted in bold type. The query processing at B_C is performed in the same way since B_C stores three POIs. As a result, P_5 is inserted into the BorderResult of B_C and the query processing at B_C is finished. The result of query processing at B_C is shown in Table 3.

Table 2. Result of Island Index Searching from Expansion Border Points After Retrieving Island Index of B_A

Expansion Border Points	BorderResult		
	k_1	k_2	k_3
B_A	(**P_4, 3**)	(**P_3, 4**)	(**P_1, 7**)
B_C	(P_1, 5)	(P_2, 10)	(**P_4, 11**)
B_E	(P_1, 3)	(P_2, 12)	(**P_4, 13**)

Table 3. Result of Island Index Searching from Expansion Border Points After Retrieving Island Index of B_C

Expansion Border Points	BorderResult		
	k_1	k_2	k_3
B_A	(P_4, 3)	(P_3, 4)	(P_1, 7)
B_C	(P_1, 5)	(**P_5, 9**)	(**P_2, 10**)
B_E	(P_1, 3)	(P_2, 12)	(P_4, 13)

Meanwhile, when an Island index includes less POIs than k, ou algorithm obtains all POIs by accessing the Island index of a node. Then it updates the BorderResult of all expansion border points. If the node needs to find more POIs to get the k closest POIs, an network expansion outside the query region is required at this node. Therefore, our algorithm finds expansion candidates that are adjacent to the node and outside the query region, and then arranges them in the increasing order of network distances from the node. The distances between POIs and expansion border points are calculated in the same way as the first case. For example, the Island index of B_E contains only two POIs, i.e., P_5 and P_1, whose number is less than $k = 3$ (in Table 2). Therefore, our algorithm updates the BorderResult of all expansion border points for P_5 and P_1. However, only the BorderResult of B_E is considered in the example because the accurate POIs have already been found for B_A and B_C. As a result, P_5 is inserted into the BorderResult of B_E with its distance=1. To expand from B_E, our algorithm finds expansion candidates, i.e., node H and I, that are adjacent to B_E and outside the query region. Table 4 shows the result after retrieving the Island index of B_E. The query processing through network expansions will be addressed in step 3.

Table 4. Result of Island Index Searching from Expansion Border Points After Retrieving Island Index of B_E

Expansion Border Points	BorderResult		
	k_1	k_2	k_3
B_A	(P_4, 3)	(P_3, 4)	(P_1, 7)
B_C	(P_1, 5)	(P_5, 9)	(P_2, 10)
B_E	(P_1, 3)	(**P_1, 3**)	(**P_2, 12**)

4.3 Step 3: POI Retrieval Through Network Expansion

If an expansion border point has less POIs in its Island index than the requested k value, it is necessary to find more POIs through network expansion. This step is performed as follows. First, we should check whether or not its distance from a node (V_{Selected}) which is the closest to the expansion border point is shorter than the distance of the k-th POI for the expansion border point. If so, our algorithm should perform the network expansion from the corresponding node and sets *exp_dist* as a distance between V_{Selected} and the expansion border point. Secondly, it searches POIs from the Island index of V_{Selected} and updates the BorderResult of all expansion border points. Then it finds nodes that are adjacent to V_{Selected} and outside the query region, and arranges them in the increasing order of network distance from the expansion border point. At this time,

our algorithm calculates the distances of the found POIs by adding *exp_dist* and $dist_p$ where $dist_p$ is the distances from the found POI to V_{Selected}. The calculated values are used for distance bounds for the found POIs. That is, if the shorter route to any POI is found through further network expansion, the distance to the POI is updated. This process is repeated until the distance to the closest node among expansion candidates is not shorter than that of the k-th closest POI in BorderResult. Finally, if there exists a node with shorter distance than that of the k-th closest POI, our algorithm terminates the query processing at this expansion border point. Then, it merges all POIs stored in the BorderResult being acquired through the step 1 to step 3 and returns the final result to the anonymizer.

For example, we assume that there exists an expansion border point B_E whose expansion candidates are nodes H and I. The Island indices of the nodes are shown in Fig.2. Because H has the shortest distance, it is selected for the network expansion and *exp_dist* is set as the distance from H to B_E, i.e., 4. Because *exp_dist* is shorter than the distance of the third closest POI of B_E (i.e., 12) in Table 4, it can be expanded to node H. Next, our algorithm searches the Island index of H and calculates the distances between B_E to POIs stored in the Island index of H. The distances are the summation of *exp_dist* and $dist_p$. That is, the distance to P_5 is $4+3$ and the distance to P_6 is $4+7$. Because the P_1's new distance (i.e., 7) to B_E is larger than that of the BorderResult of B_E, we do not need to update the distance. Meanwhile, P_6 is inserted into the BorderResult of B_E because the P_6's new distance to B_E (i.e., 11) is shorter than that of the current third closest POI, i.e., P_2. Table 5 shows the result after searching the Island index of node H. Because the distance to node I (i.e., 6) is shorter than that of the third closest POI (i.e., 11), our algorithm performs network expansion to the node I and *exp_dist* is set as the distance from B_E to I, i.e., 6. Then, it searches the Island index of I and calculates the distances between B_E and POIs within the Island radius of I. The distance to P_5 is $6+7$ and the distance to P_1 is $6+9$. Because both distances are larger than that of the current third closest POI of B_E, there is no update. Finally, since there are no more additional expansion candidates, the query processing at B_E is finished and the retrieved POIs are returned to the anonymizer as the k-NN query result.

Table 5. Result of Network Expansion

Expansion Border Points	BorderResult		
	k_1	k_2	k_3
B_A	(P_4, 3)	(P_3, 4)	(P_1, 7)
B_C	(P_1, 5)	(P_5, 9)	(P_2, 10)
B_E	(P_5, 1)	(P_1, 3)	(P_5, 11)

Now, we describe our IB_kRNN algorithm as shown in Fig.3. First, it loads the Island index created by preprocessing into the main memory (line 1). Secondly, it retrieves all POIs located on road segments that form a query region R (lines 2∼4), and determines a set of expansion border points (line 5). Thirdly, it retrieves POIs by accessing the Island index of expansion border points (lines 6∼7). Fourthly, if the Island index contains the same or more POIs than k, it obtains k closest POIs by accessing the Island index. Then it updates the BorderResult of all expansion border points and terminates the query processing at the expansion border point (lines 8∼10). Otherwise, it gets all POIs by accessing the Island index. Then it updates the BorderResult of all expansion border points. In addition, it finds expansion candidates that are adjacent to the expansion border point and outside the query region, and calculates a distance (*exp_dist*) from the expansion border point (lines 11∼16). Fifthly, if a node with the smallest *exp_dist* is closer from the expansion border

```
Input: k//the number of POIs a user wants to find
       R = {seg_1, seg_2, ..., seg_n}   //query region
Output: Result   //a set of result POIs
Island Based k-Range Nearest Neighbor Algorithm
1.   loadtoMemory(Island_file);
2.   For (each segment in R)
3.      pois = InsideSearch(seg);
4.   Result = PutResult (pois);
5.   border = SetBorderPoints();
6.   For (each border)
7.      pois = IslandSearch(border);
8.      if (num(pois) ⩾ k)
9.         Result = PutResult(pois);
10.        border.BorderResult = PutResult(pois);
11.     else
12.        border.BorderResult =PutResult(pois);
13.        For (i ⩽ degree of border)
                //degree:number of expansion candidates
14.           if (border.adjnode[i] ∉ nodes inside R)
15.              Cand=InsertAdjNode(border.adjnode[i]);
16.              Cand.Set_exp_dist (border.adjnode[i]);
17.  while (Cand[0].exp_dist
              <Dist(border.BorderResult.poi[k − 1])
18.     pois =IslandSearch(Cand [0]);
19.     border.BorderResult =PutResult(pois);
20.     For (i ⩽ degree of Cand[0])
21.        if (Cand[0].adjnode[i] is not searched yet)
22.           Cand=InsertAdjNode(Cand[0].adjnode[i]);
23.           Cand.Set_exp_dist (Cand[0].adjnode[i]);
24.     Cand[0] = Cand[0].nextNode;
25.  for (each border that performed expansion)
26.     Result =PutResult(border.BorderResult);
27.  return Result;
End Algorithm
```

Fig.3. IB_kRNN algorithm.

point than from the k-th POI, the network expansion to the node is performed (line 17). Sixthly, it searches POIs from the Island index of the node and updates all BorderResult of expansion border points. Then it finds other nodes that are adjacent to the node and outside the query region (lines 18∼23). Seventhly, it selects the next closest expansion candidate and repeats the process until the algorithm cannot find other nodes (line 24). Finally, it inserts all POIs in the BorderResult into a result set (lines 25∼26) and returns the result POIs to the anonymizer (line 27).

5 Adaptive Island Index Based Query Processing Algorithm

Our IB_kRNN algorithm has the following advantages. First, it reduces the number of network expansions by pre-computing the distances between nodes and their nearby POIs. Secondly, it can rapidly process a k-NN query over a query region by using the Island index residing in main memory. However, our IB_kRNN cannot consider POI density of each node because it uses the same radius for all nodes when generating the Island index. The shorter radius can be used for nodes with higher POI density. Based on the concept, we propose an adaptive IB_kRNN (AIB_kRNN) algorithm. Our AIB_kRNN uses an adaptive Island index which is generated by considering the POI density of each node. So, our AIB_kRNN adaptively applies different radius to each node and performs k-NN query processing over a query region by using the adaptive Island index. Our AIB_kRNN has the following advantages. In case of nodes with high POI density, the number of POIs stored in the adaptive Island index can be reduced by using short radius, thus leading to good performance on storage space usage. Meanwhile, in case of nodes with low POI density, the number of POI in the Island index can be increased by using long radius. By doing this, the number of network expansions can be decreased, resulting in good query processing performance.

Our adaptive Island index is created as follows. At first, our AIB_kRNN algorithm calculates the POI density of each node. For this, we assume that *MINk* is the number of POIs to be stored in the Island index of each node. Our algorithm finds *MINk* POIs from each node through a network expansion and sets the distance between a node and its *MINk*-th POI as *dist_to_K*. In addition, it sets the total and the average length of the retrieved road segments as $DIST_{\text{total}}$ and $DIST_{\text{avg}}$, respectively. The average number of adjacent nodes for the retrieved road segments is defined as *Degree*. Thus the POI density for each node is calculated using (1).

$$Density = DIST_{\text{total}}/MINk. \tag{1}$$

When we assume that *MAXk* is the maximum POIs that a user wants to obtain for a query, the length of road segments to be retrieved for finding ($MAXk - MINk$) additional POIs for each node, i.e., $DIST_{\text{add}}$, is calculated by using:

$$DIST_{\text{add}} = Density \times (MAXk - MINk).$$

To use $DIST_{\text{add}}$ for generating an adaptive Island index, it is necessary to transform $DIST_{\text{add}}$ into a radius. For this, we assume a virtual road network where the number of adjacent nodes is *Degree* and the length of road segments is $DIST_{\text{avg}}$when finding *MINk* POIs. In case the degree of a node is 2, the total length of road segments to be retrieved when expanding n hops from a node, i.e., T, is calculated by (2) by following an arithmetic progression. In case where the average degree of a node is greater than 2, the total length of road segments to be retrieved is calculated by (3) by following a geometric progression. We do not need to consider a node whose average degree is less than 2 because the node can be found only if it is atomic in road networks. So we assume the minimum value of *Degree* is 2.

$$T = 2 \times DIST_{\text{avg}} \times n, \tag{2}$$

$$T = \frac{Degree \times DIST_{\text{avg}} \times ((Degree-1)^n - 1)}{(Degree-1)-1}. \tag{3}$$

Based on these formulas, it is possible to find out the number of hops for the $DIST_{\text{add}}$ length of road segments. Table 6 shows how to calculate the number of the network expansions, i.e., hops, in both cases.

Table 6. Formulas for Calculating the Number of the Required Network Expansion

Condition	Number of Network Expansion (Hops) Required
$Degree = 2$	$n = \dfrac{DIST_{\text{add}}}{2 \times Dist_{\text{avg}}}$
$Degree \geqslant 3$	$n = \log_{(Degree-1)} \left(\dfrac{DIST_{\text{add}} \times (Degree-2)}{Degree \times DIST_{\text{avg}}} + 1 \right)$

Once the number of the network expansion required for each node is computed, our algorithm calculates a radius for each node to generate our adaptive Island index. The radius is calculated by adding the length of road segments for finding both *MIMk* POIs and the additional $MAXk - MINk$ POIs. Meanwhile, because this is based on the virtual road network, we can adjust the radius (*radius*) by inserting an error rate (i.e., α) as shown in (4). Finally, we generate our adaptive Island index for a node by using the calculated radius. Our AIB_kRNN algorithm is identical to our IB_kRNN algorithm, except using the adaptive Island index. Hence,

we skip the detailed explanation of our AIB_kRNN algorithm.

$$radius = dist\_to\_K + (1 + \alpha) \times (n \times DIST_{\text{avg}}). \quad (4)$$

6 Performance Analysis

In this section, we first present the environment of our performance analysis and then compare the performances of our k-NN query processing algorithms with that of an existing scheme, kRNN.

6.1 Experimental Environment Setting

We implement our k-NN query processing algorithms under the environment setting as shown in Table 7. We use the real road map of San Francisco (600 km^2) which consists of 223 200 road segments and 175 344 nodes. POI datasets are generated by using the Network-Based generator[15] on the San Francisco map data. We also use some variables for our experiments as shown in Table 8. The POI density means the proportion of the number of POIs used in the experiments to the total number of road segments in the map data, i.e., 22 026 POIs for the case of $Density = 0.1$. In addition, we adopt the XStar scheme[10] to generate query regions. This is because to the best of our knowledge, the XStar scheme is the only work that creates a cloaking area in road networks. Each query region consists of a set of the connected road segments and we generate 50 queries with different l-diversity for each experiment.

Table 7. Environment Setting

Item	Capacity
CPU	Intel Pentium® Dual-Core E6600 3.06 GHz
Memory	2 GB
OS	Windows XP professional
Compiler	Microsoft Visual Studio 2005

Table 8. Variables

Parameter	Range	Default
POI density	0.01, 0.02, 0.05, 0.1	0.1
Requested k	1, 5, 10, 15, 20	10.0
MINk	–	5.0

We compare our algorithms against kRNN (k-Range Nearest Neighbor)[7] because kRNN shows the best performance among the existing algorithms to process over a query region in road networks. We compare the performances of our algorithms and that of kRNN, in terms of the size of the generated index, the number of POIs retrieved, and query processing time. Here, the number of POIs indicates the summation of the number of POIs retrieved in road segments inside a query region and the number of POIs in our Island index. In addition, the radius for generating our Island index in IB_kRNN is set as 1% of the overall map since it was shown that the value provides the best performance[9].

6.2 Experimental Result

Fig.4 shows the size of our Island indices according to the POI density in the map. Our AIB_kRNN algorithm generates the bigger Island index than that of IB_kRNN, except the case where the number of POIs is the greatest in our experiments, i.e., POI density = 0.1. In case where the POI density is 0.1, nodes located in a high density area stores lots of POIs in IB_kRNN, whereas AIB_kRNN allows them to maintain less POIs by using an adaptively calculated radius. Meanwhile, in case where the POI density is ranged from 0.01 to 0.05, IB_kRNN stores less POIs than AIB_kRNN. The reason is that AIB_kRNN can enlarge the radius of the nodes to store enough number of POIs by considering the POI density of the region. Through our performance analysis, it is shown that the nodes located in a low density area for IB_kRNN store just a few POIs that are much less than the requested k. Because the size of the pre-processed information about the shortest paths for all nodes in kRNN is about 350 GB, which is quite huge as compared with the size of our Island indices, thus depicting only our Island indices for our both algorithms.

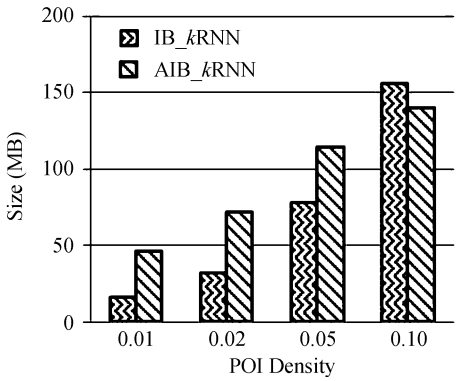


Fig.4. Size of our Island indices.

Fig.5 shows the average number of POIs retrieved when varying the required k in a logarithm way. All algorithms require more POIs when the required k is large. This is because the POIs have to be retrieved till they are further from the query region with the larger value of k. kRNN shows the worst performance because it needs to perform the network expansion to meet a road segment where the k-th POI is actually located. Meanwhile, our algorithms show better performance because they require network expand only to meet a node that includes the k-th POI inside the radius

of the node by using our Island indices. It is noteworthy that AIB_kRNN shows a bit better performance than IB_kRNN. This is because AIB_kRNN generates the adaptive Island index by considering the POI density of each node. That is, AIB_kRNN reduces the number of POIs for a node being located in a high density area, but stores more POIs for a node in a low density area. Consequently, AIB_kRNN can reduce the number of network expansions, thus leading to the decrease of the number of POIs retrieved.

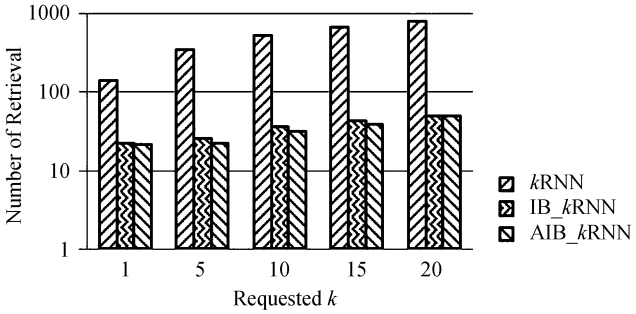


Fig.5. Number of POIs retrieved with varying k.

Fig.6 shows the average query processing time by varying the required k. All algorithms require more query processing time when the required k is higher. This is because the number of POIs retrieved is increased as the required k increases (Fig.5). Our algorithms outperform the kRNN algorithm because kRNN needs a large number of network expansions which causes high disk I/Os, whereas our algorithms make use of in-memory Island indices.

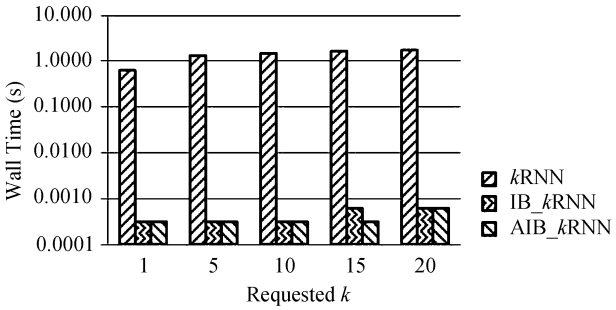


Fig.6.Query processing time with varying k.

Fig.7 shows the average number of POIs retrieved by varying the size of a query region. Here l-diversity means the average number of road segments that construct the query region. All algorithms need more POIs retrieved as the size of query region grows. This is because the number of road segments inside the query region and the number of expansion border points outside the query region are increased as the size of query region increases. However, our algorithms outperform kRNN because they use our efficient Island indices, whereas kRNN needs to do network expansion to meet a road segment where the k-th POI is actually located.

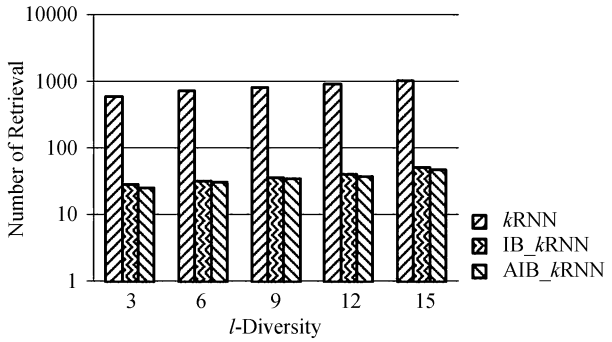


Fig.7. Number of POIs retrieved with varying the size of a query region.

Fig.8 shows the average query processing time by varying the size of a query region. Our algorithms outperform the existing kRNN algorithm in all cases. The reason is that our algorithms need less POIs retrieved as well as make use of our Island indices residing in the main memory. Because kRNN causes a large number of disk I/Os to gain accesses to a pre-computed index for distances among nodes, it shows the worst performance.

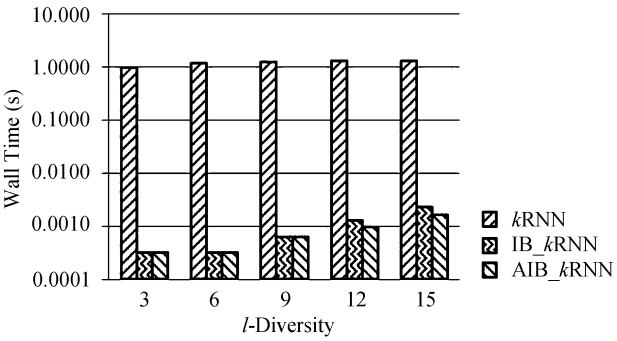


Fig.8. Query processing time with varying size of a query region.

Fig.9 shows the average number of POIs retrieved with varying POI density. All algorithms need more POIs retrieved when the POI density is higher. This is because there exist just a few POIs when the POI density is low. Therefore, more network expansions outside the query region are required to find the requested k number of POIs. Our algorithms outperform kRNN because they use our Island indices to find POIs earlier; whereas kRNN needs to do network expansion until meeting a road segment that the k-th POI is actually located on. Because our AIB_kRNN constructs the Island index in an adaptive way, it shows better performance than IB_kRNN. AIB_kRNN is much better than IB_kRNN when the POI density is low.

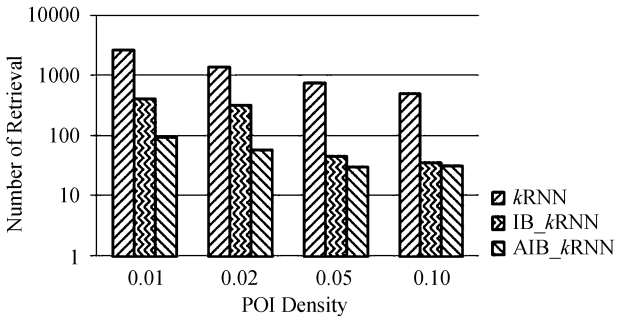


Fig.9. Number of POI retrieved with varying POI density.

Fig.10 shows the average query processing time with varying POI density. The query processing time of all the algorithms is increased as the POI density decreases. This is because the number of POIs retrieved is increased as the POI density decreases as shown in Fig.7. Our algorithms outperform kRNN in all cases because they need less POIs retrieved and use our Island indices residing in the main memory. Because the existing kRNN algorithm causes a large number of disk I/Os to gain accesses to a pre-computed index for distances among nodes, it shows the worst performance. Our algorithms outperform kRNN because they make use of our Island indices to reduce the number of POIs retrieved by avoiding disk I/O accesses.

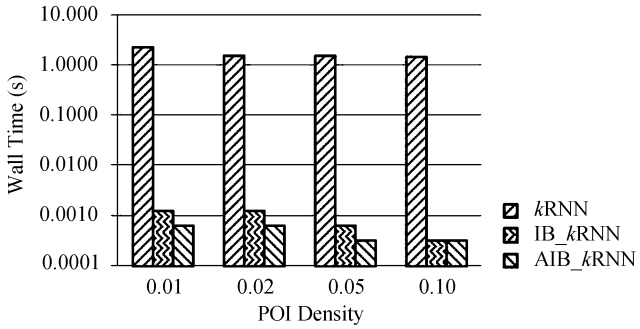


Fig.10. Query processing time with varying POI density.

Although our algorithms show better performance than kRNN, there is a gap between the query processing performances of our two algorithms when the POI density is low. Therefore, we can see that the number of POIs retrieved has a great effect on the query processing performance even though our algorithms perform k-NN query processing by using our in-memory indices. Therefore, it is more reasonable to use AIB_kRNN because it can reduce the number of POIs retrieved by using the adaptively constructed Island index.

7 Conclusions

Recently, many LBS applications have been rapidly diffused with the development of wireless communication technologies and the popularity of smart phones. However, a user's private information is in danger because the exact location of the user has to be sent to an LBS server to enjoy location-based services. Because cloaking area creation schemes have been actively studied to protect the user's location privacy, it is necessary to perform k-NN query processing over a query region.

In this paper, we proposed two k-NN query processing algorithms over a query region in road networks, i.e., IB_kRNN and AIB_kRNN. They make use of our Island indices to reduce the number of POIs retrieved. We also store our Island indices in a main memory to support fast query processing. Because AIB_kRNN adaptively generates the Island index by considering the POI density of each node, it guarantees the efficient use of storage and the reduction of the number of POIs retrieved. We showed through our performance analysis that our algorithms outperform the existing kRNN algorithm in terms of the number of POI retrieved and the query processing time.

As future work, we need to study on a continuous query processing algorithm over a query region in road networks, so as to support queries that a user continuously requests.

References

[1] Warrior J, McHenry E, McGee K. They know where you are. *IEEE Spectrum*, 2003, 40(7): 20-25.

[2] Voelcker J. Stalked by satellite: An alarming rise in GPS-enabled harassment. *IEEE Spectrum*, 2006, 43(7): 15-16.

[3] Mokbel M F, Chow C, Aref W G. The new casper: Query processing for location services without compromising privacy. In *Proc. the 32nd Int. Conf. Very Large Data Bases*, September 2006, pp. 763-774.

[4] Ghinita G, Kalnis P, Skiadopoulos S. MOBIHIDE: A mobilea peer-to-peer system for anonymous location-based queries. In *Proc. the 10th Int. Symposium on Advances in Spatial and Temporal Databases*, July 2007, pp. 221-238.

[5] Kim H, Shin Y, Chang J. A grid-based cloaking scheme for continuous queries in distributed systems. In *Proc. the 11th Int. Computer and Information Technology*, August 2011, pp. 75-82.

[6] Lee H, Oh B, Kim H, Chang J. Grid-based cloaking area creation scheme supporting continuous location-based services. In *Proc. the 27th ACM Symposium on Applied Computing*, March 2012, pp. 537-543.

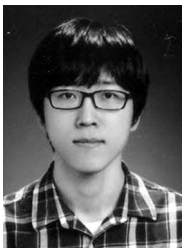
[7] Ku W, Chen Y, Zimmermann R. Privacy protected spatial query processing for advanced location based services. *Wireless Personal Communications*, 2009, 51(1): 53-65.

[8] Bao J, Chow C, Mokbel M F, Ku W. Efficient evaluation of k-range nearest neighbor queries in road networks. In *Proc. the 11th Int. Conf. Mobile Data Management*, May 2010, pp. 115-124.

[9] Huang X, Jensen C S, Šltenis S. The islands approach to nearest neighbor querying in spatial networks. In *Proc. the 9th Int. Symposium on Advances in Spatial and Temporal Databases*, August 2005, pp. 73-90.

[10] Wang T, Liu L. Privacy-aware mobile services over road networks. *PVLDB*, 2009, 2(1): 1042-1053.

[11] Kalnis P, Ghinita G, Mouratidis K, Papadias D. Preventing location-based identity inference in anonymous spatial queries. *IEEE Transactions on Knowledge and Data Engineering*, 2007, 19(12): 1719-1733.
[12] Chow C, Mokbel M F, Naps J, Nath S. Approximate evaluation of range nearest neighbor queries with quality guarantee. In *Proc. the 11th Int. Symposium on Advances in Spatial and Temporal Databases*, July 2009, pp.283-301.
[13] Xu J, Tang X, Hu H, Du J. Privacy-conscious location-based queries in mobile environments. *IEEE Transactions on Parallel and Distributed Systems*, 2010, 21(3): 313-326.
[14] Um J, Kim Y, Lee H, Jang M, Chang J. k-nearest neighbor query processing algorithm for cloaking regions towards user privacy protection in location-based services. *Journal of Systems Architecture*, 2012, 58(9): 354-371.
[15] Brinkhoff T. A framework for generating network-based moving objects. *GeoInformatica*, 2002, 6(2): 153-180.



Hyeong-Il Kim received the B.S and M.S degrees in computer engineering from Chonbuk National University, Korea, in 2009 and 2011 respectively. He is currently in a Ph.D. course in Chonbuk National University. His research interests include security and privacy of database, query processing algorithm, and cloud computing.

Jae-Woo Chang received the B.S. degree in computer engineering from Seoul National University, Korea, in 1984, the M.S. and Ph.D. degrees in computer engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 1986 and 1991, respectively. During 1996∼1997, he stayed in University of Minnesota as a visiting scholar. And during 2003∼2004, he worked for Pennsylvania State University (PSU) as a visiting professor. He joined the faculty of the Department of Computer Engineering at Chonbuk National University in 1991. Currently, he is a program committee of the IEEE Mobile Services (MS), Database Systems for Advanced Applications (DASFAA), and Asia-Pacific Web Conference (APWeb). His research interests include spatial network database, context awareness and storage system.