PAPER

A Privacy Protected k-NN Query Processing Algorithm Based on Network Voronoi Diagram in Spatial Networks

Jung-Ho UM[†], **Miyoung JANG**[††], ***Nonmembers***, ***and*** **Jae-Woo CHANG**[††a)], ***Member***

SUMMARY With the advances in wireless Internet and mobile positioning technology, location-based services (LBSs) have become popular. In LBSs, users must send their exact locations in order to use the services, but they may be subject to several privacy threats. To solve this problem, query processing algorithms based on a cloaking method have been proposed. The algorithms use spatial cloaking methods to blur the user's exact location in a region satisfying the required privacy threshold (k). With the cloaked region, an LBS server can execute a spatial query processing algorithm preserving their privacy. However, the existing algorithms cannot provide good query processing performance. To resolve this problem, we, in this paper, propose a k-NN query processing algorithm based on network Voronoi diagram for spatial networks. Therefore, our algorithm can reduce network expansion overhead and share the information of the expanded road network. In order to demonstrate the efficiency of our algorithms, we have conducted extensive performance evaluations. The results show that our algorithm achieves better performance on retrieval time than the existing algorithms, such as PSNN and kRNN. This is because our k-NN query processing algorithm can greatly reduce a network expansion cost for retrieving k POIs.

key words: *location based services (LBS), cloaking region based query processing algorithm, k-NN query processing algorithm, Network Voronoi diagram*

1. Introduction

Recently, location-based services (LBSs) have been popular due to their importance in our daily lives, services like Telematics, Intelligent Transport Systems and kiosks. LBSs provide location information that a user wants based on his/her current position. Examples of location information are gas stations or restaurants, which are commonly known as Points of Interest (POIs). The user gains such information by forwarding a query to an LBS server along with his exact location. The LBS server sends some POI information to answer the user's request. However, the user's exact location can be overheard by adversaries and the user's private information related to his behavior can be revealed [4], [5]. Using the location information, an adversary can track the user's personal habits, and places where he/she visits regularly. Therefore, the user's location must be protected while using LBS applications.

To resolve this problem, many researchers have proposed spatial cloaking algorithms to protect a user's privacy [2], [3], [5], [6], [8]. We define spatial cloaking as for a given user's request, the LBS system generates as small a query region to satisfy k-anonymity as possible. The k-anonymity means that the generated region contains a query issuer along with k-1 other individuals so that the query issuer is indistinct. Thus, to perform privacy preserving query processing, the LBS system should search k nearest POIs for a query region, rather than a query point.

Meanwhile, some recent studies have been done on searching k-nearest POIs for the cloaking region to preserve a user's location privacy [5]–[7]. However, because they only consider Euclidean space for k-nearest POIs retrieval, it is not appropriate when applying them to road network based LBSs. Furthermore, most existing query processing algorithms that consider spatial networks cannot protect users' locations because they answer a user's query for only a query point rather than a cloaking region. To overcome this problem, W. Ku et al. [8] proposed k-NN query processing algorithms for a cloaking region on spatial networks, i.e., a privacy protected spatial network nearest neighbor (PSNN) query. Their k-NN query processing algorithm finds k POIs for a cloaking region by expanding the network from intersection points between the cloaking region and road segments. However, the PSNN algorithm has two main disadvantages. First, the time complexity of the PSNN algorithm is increased as the number of road segments increases. Second, the algorithm is inefficient because the same road segments might be expanded repeatedly while expanding the network from all intersection points.

To resolve these problems, we propose a novel privacy-protected k-NN query processing algorithm using a Voronoi diagram on spatial networks. First, our algorithm reduces the network expansion cost for retrieving k-NN POIs because it finds neighboring POIs efficiently by using a network Voronoi diagram. Second, our algorithm avoids the repeated expansion of the same road segment because our algorithm can share all the intersection points created by a cloaking region while expanding the network.

The rest of the paper is organized as follows. In Sect. 2, we present related work. In Sect. 3, we describe a new k-NN query processing algorithm using a network Voronoi diagram and we provide the analytic model of our k-NN query processing algorithm. In Sect. 4, we show the performance analysis of our query processing algorithm. Finally, we conclude our work with future research directions in Sect. 5.

Manuscript received November 2, 2012.
Manuscript revised July 12, 2013.

[†]The author is with Korea Institute of Science and Technology Information, Daejeon, 305–806, South Korea.
[††]The authors are with the Dept. of Computer Eng., Chonbuk National Univ., Jeonju, 561–756, South Korea.
a) E-mail: jwchang@jbnu.ac.kr (Corresponding author)

DOI: 10.1587/transinf.E97.D.1735

2. Related Work

For LBSs, k-NN query processing on road networks has been widely studied [10]–[15]. However, the traditional approach has mainly focused on query processing for a query point, not for a query region. Some recent studies dealt with k-NN query processing based on a cloaking region in road networks. In the studies, the exact location of a kNN query issuer was blurred into a spatial region for preserving privacy. These previous studies can be categorized into three groups; i) expansion-based query processing, ii) local pre-computation based query processing methods, and iii) global pre-computation based query processing.

For the expansion-based k-NN query processing, W. Ku and Y. Chen first proposed a k-NN query processing algorithm for a cloaking region on spatial networks, namely, the privacy protected spatial network nearest neighbor query (PSNN) [10]. If the PSNN algorithm computes a cloaking region including at least k POIs, it stores these POIs into a candidate set and computes upper bounds as distances between the k-th POI and the intersection points in the region. After setting the upper bounds, the PSNN algorithm expands the network from the intersection points within the upper bounds to finds POIs. If a cloaking region contains less than k POIs, the PSNN algorithm expands the network to find more POIs until k POIs are satisfied. However, the PSNN algorithm has two main disadvantages: i) the time complexity of the PSNN algorithm increases as new road segments are found in the network expansion phase for retrieving k-NN POIs. Because the PSNN was proposed by extending the incremental network expansion algorithm, the PSNN algorithm is considered to use Dijkstra's algorithm for network distance calculation. Thus the time complexity of the PSNN is O($nlogn + c$), where n is the number of vertexes. ii) While expanding the network from all intersection points, the same road segment might be expanded repeatedly to find k POIs. As a result, the time complexity of the PSNN algorithm is proportional to O($nlogn$). Next, K. Mouratidis and M.L. Yiu proposed an anonymization technique as well as a k-NN query processing algorithm [12]. But their k-NN query processing algorithm stores the previously visited nodes to reduce the POI retrieval cost.

Second, the local pre-computation based query processing algorithm utilizes pre-computed distances among adjacent POIs. M. Kolahdouzan and C. Shahabi proposed a Voronoi-based network nearest neighbor algorithm (VN3) to process a k-NN query on spatial networks [13]. The VN3 algorithm generates a network Voronoi diagram for a given a set of POIs on the road network and pre-computes network distances between borders within each Voronoi polygon. The VN3 algorithm finds a Voronoi polygon including the given query region and retrieves its adjacent Voronoi polygons. Then, it creates an auxiliary network that consists of the border points of Voronoi polygons and POIs. By using a shortest path search algorithm, it can find the second nearest neighbor. The algorithm extends Voronoi polygons from the second nearest neighbor to find k-NN POIs. Next, X. Huang et al. proposed an S-GRID (Scalable Grid) that makes the updates of POIs more efficient by using a node pre-computation technique [14]. The S-GRID pre-computes the network distances between nodes and POIs within a grid cell by representing a spatial network in two-dimensional grids. For each grid cell, the algorithm computes the distances between a node and an edge within the cell, those between cell border points, and those between a node and a border point. The k-NN query processing algorithm of the S-GRID consists of two steps: the expansion of a grid cell containing a query point and the expansion of its adjacent cells.

Third, the query processing algorithm based on global pre-computation pre-computes all the network distances. X. Huang et al. proposed the Islands approach which pre-computes the respective "islands" of all the nodes [15]. The POIs located within a given radius from a node make an island of the POIs and the distances of all POIs from the node are recorded. At the query processing phase, the algorithm performs network expansion from the query point by using the pre-computed islands to reduce the network expansion cost. The island-based k-NN query algorithm performs as follows. Once a query is sent, the algorithm stores the vertices that are included in the query region into a heap in the ascending order of the distance to the query point. Then, from the nearest vertex to the query point, it expands the query region. When a new vertex is retrieved, the algorithm searches the POIs located within its island and this step iterates until the expansion distance is greater than the distance between a k-NN POI and the query point. Next, J. Bio et al. proposed a k-range nearest neighbor query processing algorithm (kRNN). The main idea of kRNN algorithm is to share the search execution for each boundary point of the query. The shared execution paradigm requires the shortest network distance from each boundary point to a certain set of objects to answer the query. The algorithm introduces a new tuning parameter to control the amount of space for storing the pre-computed network distance to achieve a tradeoff between the query processing performance and the storage overhead.

Table 1 shows the classification of the existing work based on the three approaches.

Table 1 Classification of the existing work.

Expansion	Local pre-computation	Global pre-computation
W. Ku and Y.Chen [10]	M.Kolahdouzan and C. Shahabi [13]	X.Huang et al. [15]
K. Mouratidis and M. L. Yiu [12]	X.Huang et al.[14]	J. Bio et al. [11]

3. System Architecture

Before we describe the system architecture for our query processing algorithm, we give the motivation behind our investigation. The dominant work for the expansion-based k-

NN query processing algorithm is the PSNN algorithm. It suffers from two main disadvantages. First, the time complexity of the PSNN algorithm increases as the number of road segments in the queue increases. Because the PSNN was proposed by extending the incremental network expansion algorithm, the PSNN algorithm is considered to use Dijkstra's algorithm for network distance calculation. So, the time complexity of the PSNN is O($nlogn + c$), where n is the number of vertexes. Second, while expanding the network from all intersection points, the same road segments might be expanded repeatedly to find k POIs for all query points. As a result, the time complexity of the PSNN algorithm is proportional to O($nlogn$).

To resolve the problems, it is necessary to adopt a pre-computation technique to k-NN query processing algorithms on spatial networks, such as VN3 [12], Island [13] and S-Grid [14]. First, The Island algorithm shows better performance on query processing time than VN3. However, as the radius increases, the Island algorithm suffers from storage overhead for storing pre-computed distances and results in a higher POI update cost. Meanwhile, the S-Grid algorithm is generally worse on query processing time than VN3 because the expansion of a grid cell depends on the number of POIs stored in the grid cell. Finally, the VN3 utilizes the Voronoi diagram to reduce the number of expansions, but the query processing performance declines if the POI density is high. Therefore, the existing algorithms cannot provide good query processing performance because they suffer from high network expansion costs.

To solve the problems of the existing approaches, we propose a k-NN query processing algorithm based on a network Voronoi diagram on spatial networks. The proposed algorithm pre-computes the distance between POIs to reduce the network expansion costs. This is because our algorithm always returns genuine k-NN POIs for a query region regardless of the actual user location within the query region.

Figure 1 depicts the system architecture for our privacy-protected query processing model. There are two main components in this system: a user and an LBS provider. The user can be an individual who uses the LBS system for searching desired POI information such as nearest gas stations, restaurants or even other users. We assume that a user generates a cloaking region either by peer-to-peer communication or by using a trusted client with a location cloaker to blur his exact location before sending a query to the server. Then, the LBS provider performs a query and returns a candidate result set to the user. Similar to the existing privacy-aware system architecture, our query processing server is embedded inside the LBS system to deal with a cloaked query region rather than the exact user's location. Therefore, the k-NN query result is given as a candidate set of POIs. After receiving the candidate set, the user evaluates his exact query result from the received candidate POI set.

Our k-NN query processing flow is explained as follows. In the pre-processing phase, an LBS provider generates a network Voronoi index on a spatial network by calculating all the distances between POIs and the network. The LBS provider first runs a network Voronoi generator to partition the network space into Voronoi polygons. Then, the LBS provider constructs a spatial index by computing the distances between POIs and network border points that are the intersections between the network edge and Voronoi polygons. At query time, a query issuer generates a cloaking region using a cloaking region creation method to hide his/her exact location. Then, the user sends a k-NN query with the cloaking region to the server. The LBS server retrieves the network Voronoi based spatial index in order to search candidate k-NN POIs, instead of the direct expansion of the network edges intersecting the query region. After this step, the server returns the candidate query result set to the user. Finally, the user selects a real query result for a query point from those of the query region.

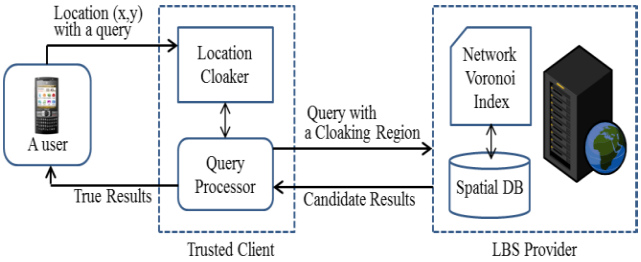


Fig. 1 System architecture.

4. Network Voronoi Diagram for k-NN Query Processing

A network Voronoi diagram is a specialization of Voronoi diagrams where the locations of POIs are restricted to the links, and a distance between POIs is defined as the length of the shortest distance on the road network. Assuming that a weighted graph G(N, L) consists of a set of nodes $N = \{n_1, n_2, \ldots, n_o\}$ and a set of links $L = \{l_1, l_2, \ldots, l_k\}$, we can generate a network Voronoi diagram for a set of POIs $P = \{p_1, p_2, \ldots, p_n\}$. For any POI $p_i \in P$, every node within the Voronoi polygon of p_i always has a shorter distance from p_i than other POIs because a Voronoi polygon contains only one POI. Therefore, for a point p on a link in L and pi in N, we define $d_n(p, pi)$ as the shortest network distance from p to pi. For all $j \in I_n|\{i\}$, we define $Dom(p_i, p_j)$ as follows. $Dom(p_i, p_j)$ denotes a set of links that are located in the Voronoi polygon of p_i where p_j can be any POI except p_i and l_o represents the set of links.

$$Dom(p_i, p_j) = \left\{ p | p \in \bigcup_{o=1}^{k} l_o, d_n(p, p_i) \le d_n(p, p_j) \right\} j \in I_n \backslash \{i\} \quad (1)$$

When $B(p_i)$ is the set of border points of the Voronoi polygon of POI p_i, any border point b in $B(p_i)$ satisfies the following condition.

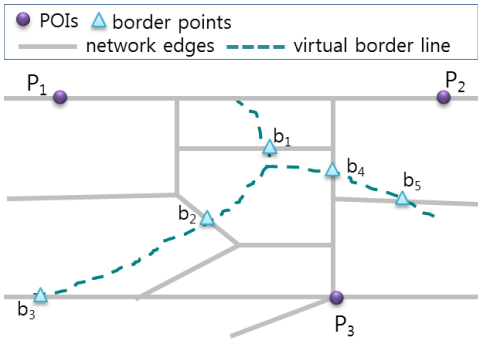


Fig. 2 Network Voronoi diagram.

$$B(p_i) = \left\{ b | b \bigcup_{o=1}^{k} l_o, d_n(p, p_i) = d_n(p, p_j) \right\} \quad (2)$$

Therefore, we define the network Voronoi diagram as follows.

Definition 1. Voronoi polygon: For any POI in a set of POIs P = $\{p_1, p_2, \ldots, p_n\}$, the Voronoi polygon is the minimum network distance between p and p_i and can be defined as follows.

$$V_{link}(P_i) = \bigcap_{j \in I_n \setminus \{i\}} Dom(p_i, p_j)$$

Definition 2. Network Voronoi diagram: If $V_{link}(p_i)$ is the Voronoi polygon of POI p_i, the network Voronoi diagram for a set of POIs P, i.e., NVD(P), is defined as follows.

$$NVD(P) = \{V_{link}(P_1), V_{link}(P_2), \ldots, V_{link}(P_n)\}$$

As shown in Fig. 2, b_1 becomes a border point for the Voronoi polygons p_1 and p_2 because b_1 is the bisector of point p_1 and p_2 when network distance is considered. In the same manner, we can generate a network Voronoi diagram by computing border points for all POIs.

For k-NN query processing, we define a query point set of the Voronoi cell where the query is issued and define the candidates of the nearest neighbor POI.

First, the definition of a query point set is given in Definition 3. A query point set can be extracted from an anonymized query region which is generated by applying a cloaking algorithm to an original query point.

Definition 3. Query point set Q:

Case 1. The cloaking region is given as a rectangle:

If the edges of the cloaking region (L1, L2, L3 and L4) intersect with z number of road networks, the intersection points on the road network, i.e., $q_1, q_2, \ldots, q_z$ are included in the query point set Q.

Case 2. The cloaking region is given as a set of road networks:

Assuming that a given set of anonymized road segments is represented as CR = $\{(s_1, e_1), (s_2, e_2), \ldots, (s_z, e_z)\}$, where s_i and e_i are the starting and ending nodes of a road segment, respectively. A query point set Q is defined as the inclusion of an edge (s_i, e_i) such that $s_i \neq s_j$ *and* $s_i \neq e_j$ *or* $e_i \neq s_j$ *and* $e_i \neq e_j$ where $s_j, e_j \in$ CR, $j \in I_n\{i\}$.

Second, our algorithm finds the NN POIs of query points by searching Voronoi cells that contain a given query point set. Therefore, the NN POI can be defined as follows.

Definition 4. Candidates of the NN POI: for a given query edge set Q, the candidates of the NN POI, i.e., *Cand*(Q), can be defined as follows. Here, Contain is a simple function to acquire POIs in the given query edge q_i, p_i is POIs included in q_i, and m is the number of NN POI candidates.

$$\begin{aligned} &\text{Cand(Q)} \\ &= \{Contain(q_1), Contain(q_2), \ldots, Contain(q_z)\} \\ &= \{p_1, p_2, \ldots, p_m\},\ m \geq 1 \end{aligned}$$

Third, if two different query edges q_i and q_j include the same NN POI, it has a high probability that we will extend the Voronoi cell of the same NN POI in the next step to find k-NN POIs. Therefore, it is necessary to group query edges that share the same Voronoi cell in order to reduce network expansion cost. We can group the query edges sharing the Voronoi cell of the same POI. To expand Voronoi cells for *k*-NN POIs, we construct Voronoi cells that include candidate POIs for NN.

The proposed query processing algorithm uses an auxiliary network (AN), which is a set of border points and query points. By using AN, we can retrieve k-NN POIs by expanding the points in the AN. The AN can be defined as follows.

Definition 5. An auxiliary network (AN): An auxiliary network can be defined as a graph $G(N_{an}, L_{an})$ where N_{an} represents border points or query points, i.e., $N_{an} = \{b_i | b \in B \text{ or } b \in Q\}$ where B is the border point set, and L_{an} represents the link between two border points or between a border point and a query point. Thus, the distance of L_{an} can be defined as:

$$L_{an} = \{d | d = d_n(b_i, b_j) \text{ where } b_i, b_j \in N_{an}\}.$$

The border points of the query region can be divided into two types: intermediate border point (IB) and end border point (EB). The intermediate border point resides on a border between Voronoi polygons of query points while the end border point includes the ones that are not intermediate points. In Fig. 3, when a query point set Q is $\{q_1, q_2\}$ and a candidate POI set Cand is $\{p_1, p_2\}$, b1 and b2 are intermediate border points and b3, b4 and b5 are end border points. Because b1 and b2 become the border points of both p1 and p2 at the same time, both b1 and b2 are included in Cand. Definitions 6 and 7 define the intermediate border point and the end border point, respectively.

Definition 6. Intermediate border point (IB): A border point b is an intermediate border point if it is constructed for two POIs, i.e., p_i and p_j, and both p_i and p_j belong to the same candidate set of POIs, i.e., Cand(Q).

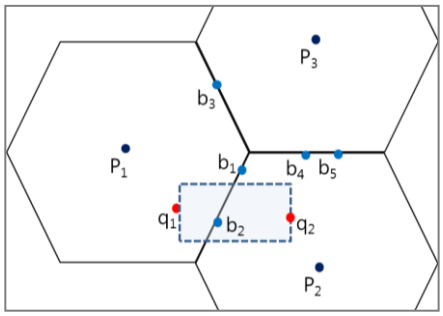


Fig. 3 Intermediate borders.

Definition 7. End border point (EB): A border point b can be defined as an end border point if it is constructed for two POIs, i.e., p_i and p_j, and either p_i or p_j belongs to a candidate set of POIs, i.e., Cand(Q).

Definition 8. Adjacent end border point (AdjB): A border point b_i belonging to Q is an adjacent end border point if the border point has at least one of the adjacent vertices of an AN, such that, AdjB is a set of adjacent vertices, i.e., AdjB = $\{ab_1, ab_2, \ldots, ab_i\}$.

A query point has the same distance to all the border points, except adjacent end border points. For example, as shown in Fig. 3, a cloaking region is located on the both polygons of p_1 and p_2. There are two query points p_1 and p_2. First of all, our algorithm adds p_1 and p_2 to candidate set. If k is not satisfied, our algorithm expands a border point between q_2 and b_4 because b_4 has the shortest distance to q_2 among border points, as shown in Fig. 3. Thus, P_3 is added to the candidate set. Nevertheless, if k is still not satisfied, the algorithm should expand the border point of b_4, thus reaching border points b_6 and b_7. Moreover, q_1 and q_2 can share the distance between b_4 and b_6 and the distance between b_4 and b_7 as shown in Fig. 3.

Therefore, our algorithm computes the shortest distance for each query point by sharing border points. That is to say, it computes the shortest distance for the adjacent end border points by subtracting the distances between the first query point and the adjacent end borders from the shortest distance to the query point. Next, it gets the distance between the next query point and the adjacent end border points. The computed shortest distance for a query point can be used for computing another query point of the query set Q. Equation (3) shows the newly computed shortest distance for a new query point, where an_i is a vertex of an AN, eb_i is the adjacent end border point, q_{prev} is the previously computed query point and q_{new} is the current query point, respectively.

$$d = d_n(q_{prev}, an_i) - d_n(q_{prev}, eb_i) + d_n(q_{new}, eb_i) \quad (3)$$

5. Privacy Protected k-NN Query Processing Algorithm

5.1 Query Processing Algorithm

Our k-NN query processing algorithm consists of four steps: initialization of a query point set Q (step 1), initialization of both the auxiliary network (AN) and expansion queue (EQ) by finding NN POI (step 2), and selection of border points for k-NN POIs (step 3) and expansion of AN (step 4). Our algorithm repeats steps 3 and step 4 until it finds k number of POIs from all query points.

***STEP 1*: Initialization of a query point set**
Since the existing cloaking algorithms only generate a region for protecting a user's location, they need to find intersection points between the cloaking region and road segments. Because a cloaking algorithm generates anonymized road segments by expanding a spatial network [15], [16], we can find query points based on the cloaking algorithm. Once the cloaking region is given, we map the cloaking region to the road network and retrieve the intersection points between the cloaking region and road network. These intersection points will be the query point set for k-NN POI retrieval.

***STEP 2*: Initialization of AN and EQ by finding NN POI**
Our algorithm finds NN POI by searching Voronoi cells that are located in the given query point set Q. Because Voronoi polygons can be indexed by R-tree, we can efficiently search the Voronoi cells including query points. AN can be constructed by retrieving the NN POI and its Voronoi cell. Once AN is initialized, intermediate border points are included in AN. If an end point b_i of POI p_i is included in Cand, the distance between b_i and a query point of a Voronoi cell of p_i can be directly calculated using the pre-computation of Voronoi diagrams [12]. The distances between b_i and the other query points are computed by using the shortest distance algorithm [18]. For example, to calculate the distance between q_2 and b_3 in Fig. 4, our algorithm computes the shortest distance between q_2 and b_3 via b_1 and the shortest distance between q_2 and b_3 via b_2. The initialized AN calculates the distances between the end border points and the query points. In Fig. 5, the gray-colored rows are selected as the shortest distance while AN is initialized.

Our algorithm also calculates the shortest distance between the NN POI and the query point. For this, our algorithm creates an expansion queue, which stores the distance between a query point and the corresponding end border point. The data structure of the expansion queue (EQ) is $\{curQ_i, dist_i, curB_i, curP_i\}$ *for* $1 < i < n$, where $curQ_i$ is the current query point, $curB_i$ is an end border point and $curP_i$ is a POI containing $curB_i$.

***STEP 3*: Selection of border points**
From the set of expansion queues, our algorithm selects a border point that has the shortest distance from the query

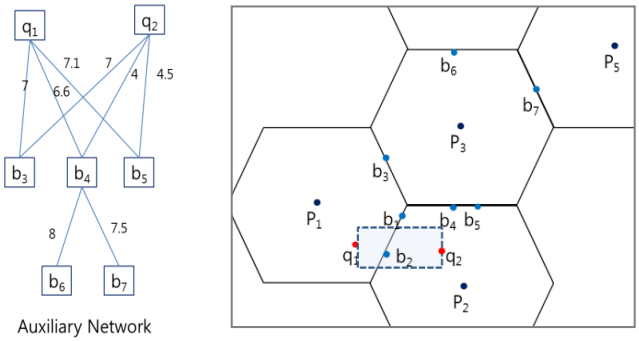
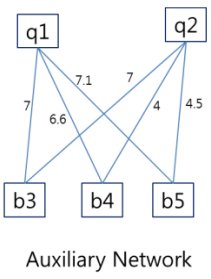


Fig. 4 Example of AN Expansion.



relation	Distance
q1-b1	3
q1-b2	2
q1-b3	7

relation	Distance
b1-b3	4
b1-b4	3.6
b1-b5	4.1

q2-b1	3
q2-b2	3.5
q2-b4	4
q2-b5	4.5

b2-b3	8
b2-b4	7
b2-b5	7.5

Network distances

Fig. 5 Example of AN initialization.

point (Eq. (4)). If more than one border points have the same shortest distance, we select the border point that shares the most query points (Eq. (5)). Here, NUM(Group) means the number of query points in a Voronoi cell and u is the number of border points that have the same distance. The border point selection continues until the set of expansion queues is empty or k-anonymity is satisfied.

$$b_{expand} = MIN(EQ(q_1).dist_1, EQ(q_2).dist_2, \ldots, EQ(q_z).dist_z) \quad (4)$$

$$b_{expand} = MAX(NUM(Group(EQ(q_i).curP_1, NUM(Group(EQ(q_{i+1}).curP_1, \ldots, NUM(Group(EQ(q_{1+u}).curP_1) \quad (5)$$

***STEP 4*: Expansion of AN**

When a border point is selected, our algorithm performs the expansion of AN by searching a Voronoi cell v_i which contains the selected border point, but is not expanded yet. Our algorithm inserts the border points for v_i, its POI and the distances between the border points and the POI into AN. Next, it computes the distances between each vertex and query points using the shortest distance algorithm. When our algorithm computes maxDist, which is the minimum expansion distance, it checks if the distance between each query point and its k-NN POI is smaller than maxDist. If so, we update the results. To share the expanded network, our algorithm stores the border points being connected directly to a query point and their distances from a query point into a main memory.

Algorithm 1. K-N Query Processing OnSpatialNetwork(k, Q)

INPUT: int k, Q *// the number of nearest neighbor, Query Points*
OUTPUT: Cand *// the candidate k-NN POIs for Q*

1. for each query point qi of Q *// initialization*
2. P=Contain(q_i)
3. Insert P into Cand(q_i, P, $d_n(q_i,P)$)
4. If k=1 then return Cand
5. Otherwise
6. for each query point q_i of Q
7. Insert qi into Group(P,q_i)
8. Find intermediate borders IB
9. for each border b_i of P
10. If bi $\in$ IB then
11. Insert P into Cand(q_i,P)
12. for each other border b_j of P, $b_j \notin$ IB
13. Insert q_i and b_j into AN(q_i,b_j, $d_n(q_i,b_i)+d_n(b_i,b_j)$)
14. Otherwise
15. Insert $d_i=d_n(q_i,b_i)$ into EQ(q_i , d_i, b_i, P)
16. Insert q_i and b_i into AN(q_i,b_j, $d_n(q_i,b_i)$)
17. $maxDist_i=\infty$
18. while EQ is not empty //expansion
19. get (d_{min} , q_i, b_i, p_i) from EQ
20. If d_{min} >= $maxDist_i$ then Cand(q_i).complete=true
21. If all Cand.complete is true then break
22. get p_j which is an adjacent POI of p_i and it has the border point b_i
23. for each border b_j of p_j
24. Insert b_i and b_j into AN(b_i,b_j,dn(b_i,b_j))
25. Set New Shortest Path of Q in AN
26. Update EQ and Cand
27. return Cand

End Algorithm

Algorithm 1 describes our k-NN query processing algorithm. Step 1 can be performed before searching the k-NN POIs according to the size or the shape of the cloaking region. First, our algorithm finds NN POI candidates for all the query points and inserts them into Cand (Lines 1–3). Second, if k = 1, our algorithm determines which group a query point belongs to based on the types of border points and retrieves the intermediate border (Lines 6–8). Third, our algorithm initializes both AN and the expansion queue (Lines 9–17). To insert a POI p_i into AN, our algorithm determines the type of border point of p_i. If it is an intermediate border point, algorithm inserts the distances between the intermediate border point and the end border points into the AN (Lines 10–13). Otherwise, the algorithm inserts the distance between the end border point and a query point into both the AN and the expansion queue (EQ) (Lines 14–16). Fourth, our algorithm initializes the distance between the current k-th POI candidate and a query point, i.e., maxDist. Then it selects a border point b_i from EQ for the expansion of AN and compares maxDist with the distance between b_i and a query point. If the distance is less than maxDist, our algorithm expands the network by acquiring a non-expanded POI p_j containing b_i and then inserts into AN both the p_j and

its border points for computing the new shortest distance (Lines 22–25). Fifth, if POIs' distances from the query point to their border points are changed due to the insertion of p_j, our algorithm updates the distances stored in both EQ and Cand (Line 26). Finally, if the expansion stops for all the query points, our algorithm returns Cand for k-NN POIs to a user (or anonymizer).

5.2 Query Execution Cost

In this section, we analyze the query processing costs of existing PSNN, kRNN algorithms and our algorithm. Table 2 shows the parameters used for our analysis.

We analyze the cost of the PSNN query processing algorithm. There are three steps in the PSNN query processing algorithm. They are: i) finding POIs (#R1) that are located in the cloaking region, ii) searching links that contain the given query points, and iii) expanding the spatial network based on the shortest distance for searching the number of k-#R1 POIs. The cost of each step is analyzed as given below. First, the cost for executing step 1 can be computed by the following equation where the POIs are indexed by R-tree and m is the number of entries stored at a node of R-tree.

$$\mathrm{PSNN}_{\mathrm{COST}_1} = O(m \times \log_{m+1}(\#P + 1)) \tag{6}$$

Second, when the links of the spatial network are indexed by R-tree, the cost for executing step 2 can be computed by using the following equation.

$$\mathrm{PSNN}_{\mathrm{COST}_2} = O(m \times \log_{m+1}(\#L + 1)) \tag{7}$$

Third, the cost for executing step 3 is the same as the execution cost of the Dijstra's algorithm, which computes the shortest distance of vertexes on spatial networks. In addition, the cost is proportional to the number of query points. Below Eq. (8) presents the cost of step 3 where c is a constant value.

$$\mathrm{PSNN}_{\mathrm{COST}_3} = O(\#Q\#N \times \log(\#N) + c) \tag{8}$$

Therefore, the cost of the PSNN query processing algorithm can be computed by combining the cost of the above three steps. Equation (9) presents the total cost of the PSNN query processing algorithm. The query processing cost of the PSNN algorithm is increased by $\#N * log(\#N)$ times with an increase of the number in query points #Q where the number of entries is m, the number of vertexes is #N, and the number of links is #L.

$$\begin{aligned}\mathrm{PSNN}_{COST} = O(&m \times (\log_{m+1}(\#P + 1) \times (\#L + 1)) \\ &+ \#Q \times \#N \times \log(\#N) + c)\end{aligned} \tag{9}$$

Next, we analyze the cost of the kRNN query processing algorithm. The kRNN has three main steps for a query processing. The first two steps are same as those of the PSNN algorithm. In the step 3, because the kRNN uses pre-computed network distances for network distance computations, the cost of the step 3 is $O(C_{dist})$. Therefore, the query processing cost of the kRNN algorithm is increased by $\#N * log(\#N)$ times with the increase of pre-computed network index retrieval time.

$$\begin{aligned}\mathrm{kRNN}_{COST} = O(&m \times (\log_F \#P \times (\#L + 1)) \\ &+ (C_{dist} \times \#N \times \log(\#N)))\end{aligned} \tag{10}$$

Finally, we analyze the cost of our k-NN query processing algorithm based on network Voronoi diagrams, called VDNN, which consists of three steps as follows: i) searching Voronoi cells containing the given cloaking region, ii) initialization of AN, and iii) expanding AN for searching the number of k POIs. When Voronoi cells are indexed by R-tree, the cost for executing step 1 is computed by the following equation.

$$\mathrm{VDNN}_{\mathrm{COST1}} = O(m \times \log_{m+1}(\#P + 1)) \tag{11}$$

In step 2, our algorithm computes the shortest distances from all query points to end border points for initialization of AN. For reducing the network expansion cost, our algorithm shares the shortest distances, which was already computed when it computed the shortest distance between other query points to end border points. Then the cost for initialization of AN can be computed by the following equation.

$$\begin{aligned}\mathrm{VDNN}_{\mathrm{COST2}} = O((&\#IB + \#EB - \#EB_{pi}) \\ &\times \log(\#IB + \#EB - \#EB_{pi}) + \#Q \\ &\times (\#IB + \#EB - \#EB_{pi}) + c)\end{aligned} \tag{12}$$

The cost of expansion of AN is similar to the cost of step 2 because only the number of border points (#B) is different in this step. Equation (12) represents the cost of this step 3.

$$\mathrm{VDNN}_{\mathrm{COST3}} = \mathrm{O}((\log(\#B) \times \#Q \times \#B) + c) \tag{13}$$

Since #IB and #EB are less than #B, the cost of step 2 can be included into the cost of step 3. As a result, the total cost of our k-NN query processing algorithm can be given by the following equation.

$$\begin{aligned}\mathrm{VDNN}_{\mathrm{COST}} = \mathrm{O}(&m \times (\log_{m+1}(\#P + 1) + \#B \times \log(\#B) \\ &+ \#Q \times \#B + c)\end{aligned} \tag{14}$$

The cost of our k-NN query processing algorithm is lower than the cost of the PSNN query processing algorithm

Table 2 Parameters used for analysis.

Parameters	Description
#N	Number of vertexes in spatial network
#L	Number of links in spatial network
#P	Number of POIs in spatial network
#Q	Number of query points in spatial network
#R1	Number of POIs in cloaking region
#B	Number of border points for a Voronoi cell
C_{dist}	Distance calculation cost in the pre-computed network distance index

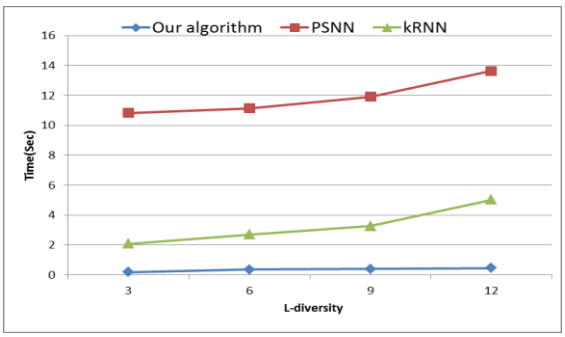
for the following reasons. First, the number of vertexes for expansion is reduced by using a Voronoi diagram in our algorithm. That is, the cost of our algorithm is reduced by 1/#n as compared to the PSNN algorithm because the number of border points #B for expansion is approximately #B = #N/#n where #n is the average number of vertexes being included in one Voronoi cell. Second, as the value of #Q increases, the cost of our k-NN query processing algorithm is increased by #B times while the cost of the PSNN algorithm is increased by #N × log(#N) times. That means that the cost of our algorithm is decreased to 1/n × log(#N), as compared to the PSNN algorithm.

6. Performance Analysis

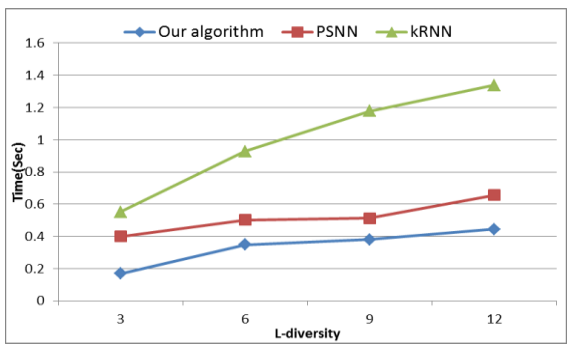
In this section, we compare the performance of our k-NN query processing algorithm with the PSNN algorithm and kRNN algorithm under different settings. We implement the algorithms using a Windows 2003 operating system with Microsoft Visual Studio 6.0 running on an Intel Xeon 3.0GHz with 2GB of RAM. For spatial network data, we use a San Francisco Bay map consisting of 220,000 edges and 170,000 nodes. We generate four sets of POIs with different densities using the Brinkhoff algorithm [18], i.e., 2,200 POIs (density = 0.01%), 4,400 POIs (density = 0.02%), 11,000 POIs (density = 0.05%), and 22,000 POIs (density = 0.1%). The density of POIs is computed as the percentage of the number of POIs over the number of links (= network edges). For the edge set of query points, we use the XStar algorithm [15], which generates an anonymized edge list. We evaluate the *k*-NN query processing time for varying *l*-diversity, the value of *k* and query window size (i.e., the percentage of the whole data area). Table 3 shows the parameters used for performance analysis.

Table 3 Experiment parameter.

Types	Values	default
l-diversity	3, 6, 9, 12	6
k	10, 20, 30, 40, 50	30
Query window	0.1%, 0.2%, 0.5%, 1.0%	0.1%



(a) POI density (0.01%)



(b) POI density (0.1%)

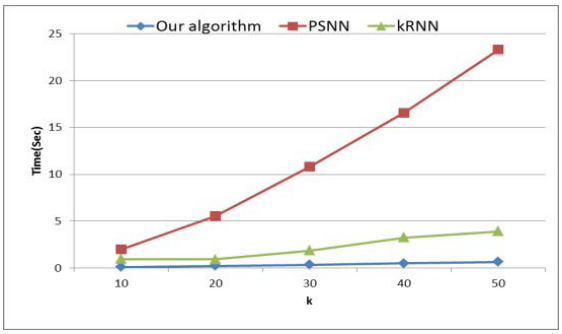
Fig. 6 k-NN query processing time with varying l-diversity.

6.1 K-NN Query Processing Time

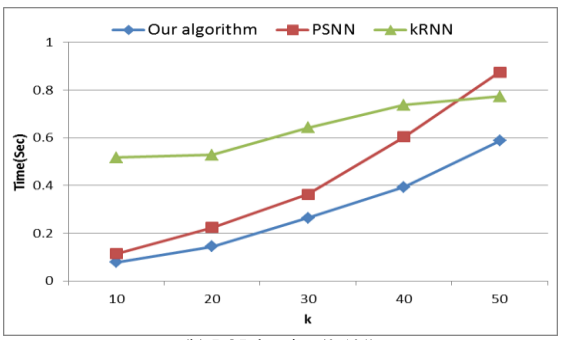
Figure 6 presents the k-NN query processing time of our algorithm (VDNN), PSNN and kRNN according to the l-diversity. As l-diversity increases, the query processing time also increases. When l-diversity is 12 and POI density is 0.1%, the query processing time for our algorithm is 0.42 sec, whereas the PSNN and kRNN required 0.64 and 1.37 seconds, respectively. The results confirm that our algorithm acquires about 2 times and 3 times performance gains on retrieval time, compared to the PSNN and kRNN, respectively. The reason is that our algorithm needs less network expansion overhead than the existing methods because it uses a network Voronoi diagram and shares the shortest distances between query points and POIs.

Figure 7 illustrates the experimental results on the k-NN query processing time when k ranges from 10 to 50. When k is 5 and POI density is 0.1%, the query processing time for our algorithm is 0.07 sec, whereas the PSNN and kRNN required 0.11 and 0.25 sec, respectively. Overall, we observe from the performance results that our algorithm is almost 1.5 times faster than the PSNN algorithm and almost 2 times faster than the kRNN algorithm. The reason is that our algorithm can greatly reduce the network expansion overhead and we can share the shortest distances between query points and POIs. Thus, we prove that our algorithm guarantees a consistent query processing performance even although k and POI density increase.

Figure 8 illustrates the impact of POI density on a k-NN retrieval time. The x axis shows the density of POI and y axis shows the time required for retrieving k-NN POIs performance when the number of retrieving nearest neighbor (k) is 30. To enhance the readability, the query processing time is depicted as a logarithm of values. As shown in the figure, both PSNN and kRNN shows better performance as the POI density is increased. This is because for more dense area, less network expansion is required for retrieving k^{th} nearest POI of each query point. On the other hand, our algorithm provides almost constant query processing performance with varying POI density. When expanding the network for k-NN POI retrieval, the existing methods calculate and compare the distances among network nodes and POIs on the fly, whereas the proposed method only retrieves the pre-computed Voronoi diagram index. This makes big

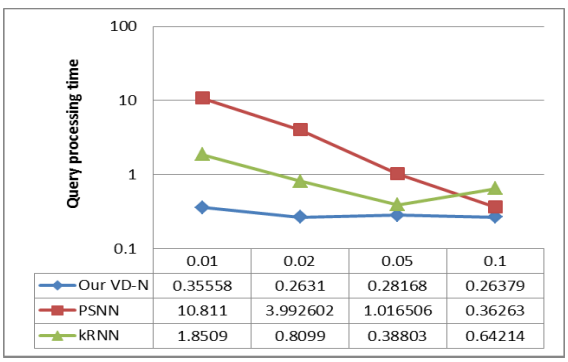


(a) POI density (0.01%)



(b) POI density (0.1%)

Fig. 7 k-NN query processing time with varying k.



	0.01	0.02	0.05	0.1
Our VD-N	0.35558	0.2631	0.28168	0.26379
PSNN	10.811	3.992602	1.016506	0.36263
kRNN	1.8509	0.8099	0.38803	0.64214

Fig. 8 k-NN query processing time with varying POI density (k = 30).

difference in overall query processing performance between the existing algorithms and the proposed algorithm. Therefore, POI density has less influence on the query performance compared to the number of k.

6.2 Updates and Storage Overhead

Because our k-NN query processing algorithm uses a network Voronoi diagram (NVD), we compare our NVD with the index structures of the PSNN and kRNN, in terms of storage overhead and insertion/deletion performances. Figures 9 and 10 show the insertion and deletion performances of our NVD, respectively. Our network Voronoi diagram requires more time for the insertion and deletion of POIs because it updates pre-computation distances whenever an insertion (or deletion) occurs.

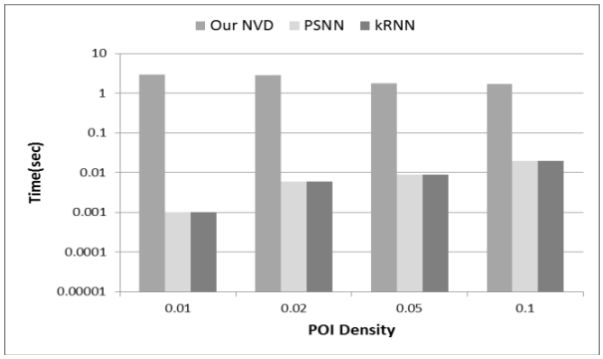


Fig. 9 POI insertion time.

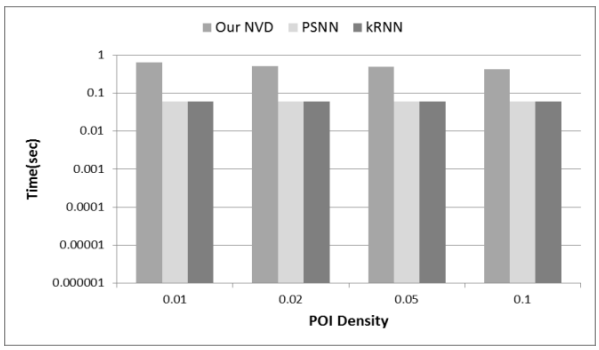


Fig. 10 POI deletion time.

Table 4 Storage overhead.

	0.01	0.02	0.05	0.1
Our NVD	468	362	273	232
PSNN	102	102	103	105
kRNN	360431	360,431	360,432	360,434

Table 4 shows the storage overhead of our NVD. Our NVD requires 2–4 times more storage than the index structure of the PSNN, whereas it requires almost 800 times less storage than that of the kRNN. This is because our NVD maintains the pre-computation information for each Voronoi cell, whereas the PSNN and kRNN do not store pre-computation information by performing network expansion at a query processing time.

6.3 Analysis

In this section, we compare the theoretical and experimental results in terms of the k-NN query processing time. For this, we set the parameters of our theoretical analysis as shown in Table 5. Here, #N is the average number of expansion nodes when performing each k-NN algorithm. In the case of the PSNN, #N is set to 6, 5, 4 and 4 because 3, 6, 9, and 12 are used as L-diversity in our experiments. The reason is that as the number of L-diversity increases, the cloaking region includes more network nodes and requires less number of network expansions. For the kRNN, #N is constant to 6 because the kRNN uses pre-computation technique. #L means

Table 5 Parameter setting.

Parameter	value
#N	PSNN:6,5,4,4; kRNN:6
#L	220,000
#P	10,000
#Q	22,36,52,65
#B	30
m	50
C_{dist}	13,26,39,52

Table 6 Theoretical model and experimental results.

		L=3	L=6	L=9	L=12
Our VDNN	Model	0.164	0.248	0.344	0.422
	experiment	0.170	0.282	0.396	0.445
	Error rate	3.35%	11.95%	13.05%	5.1%
PSNN	Model	1.081	1.313	1.307	1.620
	experiment	1.104	1.260	1.363	1.885
	error rate	2.0%	4.2%	4.11%	14.05%
kRNN	Model	0.662	1.269	1.876	2.483
	experiment	0.706	1.240	1.628	1.990
	error rate	6.28%	2.31%	15.21%	24.75%

the total number of road network and is set to 220,000 because the PSNN and the kRNN algorithms traverse all links of the R-tree index in the worst case. #P is set to 10,000 because our experiments are evaluated by using POI data with 0.05 density. #Q means the average number of query points and are set to about 20∼70 for different query window size. #B meaning the average expansion number of Voronoi cell is set to 30. The parameter *m* is set to the default number of entries in R-tree, i.e., 50. C_{dist} varies from 13 to 52 because the L-diversity is ranged from 3 to 12.

Because we assume that disk access time is 0.2ms, the error rate is calculated based on the following equation. Here, C_{exp} deontes the experimental results and C_{theroy} indicates the estimated query processing cost based on our theoretical model.

$$\frac{|C_{exp} - C_{theory}|}{C_{exp}} \times 100$$

Table 6 provides both the theoretical and experimental results of our k-NN query processing algorithm based on network Voronoi diagrams (VDNN). In addition, we show a set of relative errors between them when the l-diversity is 3, 6, 9 and 12, respectively. We can see that the error rates between the experimental and analytic results on retrieval time are 5.26% for the best case and 23.07% for the worst case. The results show that the analytic and experimental results agree very well.

7. Conclusion

In this paper, we proposed a novel k-NN processing algorithm for a cloaking region in order to protect users' location privacy in spatial networks. To reduce the number of network expansions, we used a network Voronoi diagram that stores the shortest network distance among POIs, and we changed a cloaking region into a query point set. To process queries efficiently, our algorithm shared the shortest distances from each query point to relevant POIs. Through our extensive performance analysis, we have shown that our algorithm outperforms the existing PSNN and kRNN algorithms in terms of query processing time.

As a future study, we plan to investigate how to reduce the update cost of the shortest distance stored in the auxiliary network. We also plan to extend our algorithm to process continuous k-NN queries.

Acknowledgments

This research is partly supported by the IT R&D program of MSIP/KEIT [2014044034002, High performance database solution development for Integrated big data monitoring and Analysis] and this work was supported by Basic Science Research program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (grant number 2013010099).

References

[1] D. Mohapatra and S.B. Suma, "Survey of location based wireless services," Proc. International Conference on Personal Wireless Communications, 2005, pp.358–362, 2005.
[2] A. Khoshgozaran and C. Shahabi, "Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy," Proc. International Symposium on Spatial and Temporal Databases, 2007.
[3] B. Gedik and L. Liu, "Location privacy in mobile systems: A personalized anonymization model," Proc. IEEE International Conference on Distributed Computing Systems, 2005, pp.620–629, 2005.
[4] J.H. Um, H.I. Kim, Y.H. Choi, and J.W. Chang, "A new grid-based cloaking algorithm for privacy protection in location-based services," Proc. IEEE International Conference on High Performance Computing and Communications, 2009.
[5] G. Ghinita, P. Kalnis, and S. Skiadopoulos, "MobiHide: A mobilea peer-to-peer system for anonymous location-based queries," Proc. International Symposium on Spatial and Temporal Databases, vol.4605, 2007.
[6] M.F. Mokbel, C. Chow, and W. Aref, "The new casper: Query processing for location services without compromising privacy," Proc. Very Large Data Base, 2006.
[7] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," Proc. 1st International Conference on Mobile Systems, Applications and Services, 2003.
[8] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias, "Preventing location-based identity inference in anonymous spatial queries," IEEE Trans. Knowl. Data Eng., vol.19, pp.1719–1733, 2007.
[9] M. Gruteser and D. Grunwald, "Voronoi-based K nearest neighbor search for spatial network," Proc. 13th International Conference on Very Large Data Bases, vol.30, 2004.
[10] W.S. Ku, Y. Chen, and R. Zimmermann, "Privacy protected spatial query processing for advanced location based services," Proc. Wireless Pers Commun, 2008.
[11] J. Bao, C.Y. Chow, M.F. Mokbel, and Wei-Shinn Ku, "Efficient evaluation of k-range nearest neighbor queries in road networks," Proc. Mobile Data Management, 2010.
[12] K. Mouratidis and M.L. Yiu, "Anonymous query processing in road networks," IEEE Trans. Knowl. Data Eng., vol.22, pp.2–15, 2010.

[13] M. Kolahdouzan and C. Shahabi, "Voronoi-based K nearest neighbor search for spatial network databases," Proc. Very Large Database, 2004, pp.840–851, 2004.
[14] X. Huang, C.S. Jensen, H. Lu, and S. Saltenis, "S-GRID: A versatile approach to efficient query processing in spatial networks," Proc. Symp. Spatial and Temporal Databases, LNCS 4605, pp.93–111, 2007.
[15] X. Huang, C.S. Jensen, and S. Saltenis, "The islands approach to nearest neighbor querying in spatial networks," Proc. Symp. Spatial and Temporal Databases, 2005, pp.73–90, 2005.
[16] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," Proc. Very Large Database, 2003, pp.802–813, 2003.
[17] E.W. Dijkstra, "A note on two problems in connection with graphs," Numerische Mathematik, vol.1, pp.269–271, 1959.
[18] A. Okabe, B. Boots, K. Sugihara, and S.N. Chiu, Spatial Tessellations, Concepts and Applications of Voronoi Diagrams, 2nd ed., John Wiley & Sons, 2000.
[19] T. Wang and L. Liu, "Privacy-aware mobile services over road networks," Proc. Very Large Database, 2009.
[20] http://www.fh-oow.de/institute/iapg/personen/

Jung-Ho Um is a researcher at the Korea Institute of Science and Technology Information. He received the BS, MS, and PhD degrees from Chonbuk National University in 2004, 2006 and 2010 respectively. His research interests include security and privacy of databases, spatial databases, and GIS.

Miyoung Jang is a PhD candidate in Chonbuk National University since 2011. She received the BS and MS degrees at Chonbuk National University in 2009 and 2011, respectively. Her research interests include security and privacy of databases.

Jae-Woo Chang is a professor in the Department of IT Information and Technology, Chonbuk National University, Korea since 1991. He received the BS degrees in Computer Engineering from Seoul National University in 1984. He received the MS and PhD degrees in Computer Engineering from Korea Advanced Institute of Science and Technology (KAIST) in 1986 and 1991, respectively. During 1996–1997, he was at the University of Minnesota as a visiting scholar. During 2003–2004, he worked for Penn State University (PSU) as a visiting professor. His research interests include spatial network database, context awareness and storage system.