PAPER

# TSC-IRNN: Time- and Space-Constraint In-Route Nearest Neighbor Query Processing Algorithms in Spatial Network Databases

**Yong-Ki KIM**[†a)], *Nonmember and* **Jae-Woo CHANG**[†], *Member*

**SUMMARY** Although a large number of query processing algorithms in spatial network database (SNDB) have been studied, there exists little research on route-based queries. Since moving objects move only in spatial networks, route-based queries, like in-route nearest neighbor (IRNN), are essential for Location-based Service (LBS) and Telematics applications. However, the existing IRNN query processing algorithm has a problem in that it does not consider time and space constraints. Therefore, we, in this paper, propose IRNN query processing algorithms which take both time and space constraints into consideration. Finally, we show the effectiveness of our IRNN query processing algorithms considering time and space constraints by comparing them with the existing IRNN algorithm.

***key words:*** *spatial network database, query processing algorithm, in-route nearest neighbor query, time & space constraints*

## 1. Introduction

Recently, there have been a large number of studies on query processing algorithms for coordinate-based queries and trajectory-based ones in spatial network databases (SNDB) [1]–[9]. However, there has been little work on route-based queries which find the closest POI while an object is moving on a given route. Since moving objects move on a predefined spatial network, it is essential to study efficient route-based query processing algorithms in SNDB.

Route-based queries in SNDB have been classified into two groups; continuous k-nearest neighbor query [1]–[5] and in-route nearest neighbor query [6]. There are two leading research on continuous k-nearest neighbor query. First, Kolahdouzan et al. [4] proposed a continuous k-NN query processing algorithm by using a Voronoi network diagram to calculate a network distance between two POIs (Point of Interests). Next, Mouratidis & Yie [5] proposed both an incremental monitoring algorithm and group monitoring algorithm based on R-tree. Meanwhile, one of the most important route-based queries is an in-route nearest neighbor (IRNN) query. Shekhar & Yoo [6] proposed an IRNN query to find a nearest neighbor by deviating minimally from a given planned route. However, the IRNN query processing algorithm has a problem that it cannot deal with real-time situations, like a traffic jam or a vehicle with limited petrol reserves.

Manuscript received July 9, 2010.
Manuscript revised December 22, 2010.

[†]The authors are with the Dept. of Computer Eng., Chonbuk National Univ., Chonju, Chonbuk, 561–756, Korea.
a) E-mail: ykkim@dblab.chonbuk.ac.kr

DOI: 10.1587/transinf.E94.D.1201

To solve this problem, we propose IRNN query processing algorithms considering time and space constraints. Therefore, our IRNN query processing algorithms can find the optimal deviated route to satisfy time and space constraints in real applications, such as Telematics, car navigation system (CNS), and Location-based commerce (L-commerce). In addition, we show the effectiveness of our IRNN query processing algorithms by comparing them with the existing IRNN query processing algorithm.

The rest of the paper is organized as follow. Section 2 presents related work on route-based query processing algorithms for SNDB. In Sect. 3, we propose our IRNN query processing algorithms considering time and space constraints. In Sect. 4, we provide the performances analysis of our IRNN query processing algorithms. Finally, we conclude our work with further research in Sect. 5.

## 2. Related Work

There are two leading algorithms for processing a continuous k-NN query in spatial networks. First, Kolahdouzan et al. [4] proposed a continuous k-NN query processing algorithm by using a Voronoi network diagram in spatial networks. In this algorithm, they pre-compute a network distance between POIs by using Voronoi nearest neighbor network diagram (VN3). For a given route of moving object, the VN3 first splits the given route into a set of node-to-node segments or node-to-POI segments. Then, the algorithm retrieves k nearest POIs by using the VN3 on each segment. Next, it finds a split point based on the direction of the POIs, and at the end, it retrieves k POIs by using the distance between the given query point and POIs. Secondly, Mouratidis et al. [5] proposed two continuous k-NN search algorithms by using R-tree; incremental monitoring algorithm and group monitoring algorithm. The incremental monitoring algorithm only updates objects falling in edges in the expansion tree. The group monitoring algorithm considers a path between two consecutive nodes (i.e., intersections) in the spatial network.

Meanwhile, one of the most important route-based queries is an in-route nearest neighbor (IRNN) query. Shekhar & Yoo [6] proposed an IRNN query to find a nearest neighbor by deviating from the planned route as minimally as possible, whereas a continuous k nearest neighbor query finds nearest neighbors along the shortest route. To deal with the IRNN query, they proposed four algorithms. The first one is a simple graph-based algorithm which finds the nearest neighbor from each node of a given route and then obtains candidate routes passing through each nearest neighbor. As a result set, the algorithm finds the nearest neighbor with the shortest distance on the route. However, it has a drawback that its performance is decreased as the number of node increases. The second one is a recursive spatial range algorithm which calculates distances from a current node to a nearest neighbor by using a shortest path algorithm [10]. If a new distance calculated from another node is less than the shortest distance found from now on, the new distance becomes the shortest distance. Base on the current distance, the algorithm processes a range query to find a nearest neighbor from the remaining nodes on a given route, recursively. Although the recursive range algorithm has less query processing time than the simple graph algorithm, it suffers from high processing cost because of its recursive nature. The third one is a spatial distance join algorithm which reduces the processing time of range query at each node. Since the computation of Euclidean distance from a node to a POI is faster than that of network distance, this algorithm finds a nearest neighbor by using a range query based on Euclidean distance between a node and a POI in the given route. The last one is a pre-computed zone (PCZ) algorithm which pre-computes the network distance between a node and a POI by using the service zone defined for a POI. Thus, this algorithm finds a nearest neighbor rapidly by using pre-computed distances in service zones intersecting with a given route. The PCZ algorithm has the best performance among the four algorithms in terms of query processing time because it can process the IRNN query efficiently by finding the pre-computed nearest POI from each node.

## 3. IRNN Query Processing Algorithms Considering Time and Space Constraints

For supporting such applications as Telematics, CNS, and L-commerce, it is necessary to study on IRNN query processing algorithms considering time and space constraints in spatial network databases. For this, we define IRNN queries considering time constraint, space constraint, and both time and space constraints. We also propose our IRNN query processing algorithms for the defined IRNN queries.

### 3.1 Time-Constraint IRNN Query Processing Algorithm

The existing IRNN query finds a nearest neighbor with a minimum deviation from a planned route. An example of the IRNN query is to find the nearest neighbor POI while we deviate from the planned route (R1) minimally, as shown in Fig. 1. Although routes passing through POI B or C are the shortest ones, the existing IRNN query processing algorithm returns a route R2 that passes through POI A. This is because it deviates from the given route minimally. However, the existing IRNN query processing algorithm does not

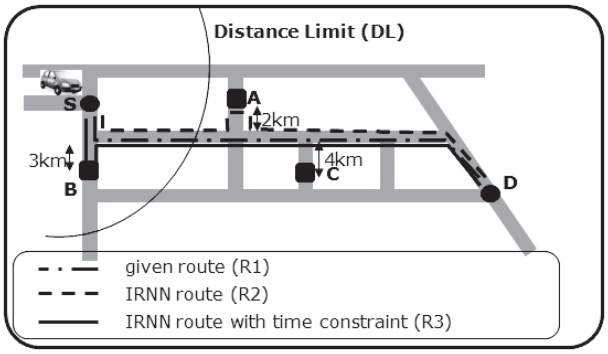

**Fig. 1** IRNN Query with time constraint.

consider time constraint. For example, in the real life, a user has to find a gas station within 20 minutes from his current position because the car does not have gas enough to travel a whole planned route. Because the existing IRNN algorithm finds the route passing through POI A, the car cannot reach to POI A due to the lack of gas. Therefore, it is necessary to find a POI which can satisfy a time constraint.

In addition, it is necessary to differentiate different types of POIs, like gas stations where the time constraint is very tight and restaurants where it is not relatively tight. For example, a user wants to find a restaurant within a given time constraint (20 minutes) by minimally deviating from the planned route. However, since the time constraint is not tight, it is possible to find a restaurant located beyond the distance limit of the time constraint. To deal with this situation, we can provide a tightness degree to indicate how tight the time constraint is. Based on the time constraint tightness, we can define a distance limit to which a car can travel. Definition 1 presents the distance limit based on the original time constraint and its tightness degree.

***Definition 1*** *We define a distance limit (DL) based on $\alpha$ where $\alpha$ is a tightness degree of time constraint.*

$$DL = (\textit{current car speed} * \textit{original time constraint})/\alpha,$$
$$(0 < \alpha \leq 1).$$

For example, we assume that the current speed of a car is 60 km/h, the original time constraint is 20 minutes, and the time constraint is extremely tight ($\alpha = 1$). In this case, the distance limit is $(60\,\text{km/h} * 20\,\text{min})/1 = 20\,\text{km}$. Whereas, if $\alpha$ equals 0.5, the distance limit is changed into $(60\,\text{km/h} * 20\,\text{min})/0.5 = 40\,\text{km}$. This means that a restaurant within 40 km from the current position can be considered as a candidate POI in case the time constraint is not tight like a restaurant. By using the tightness degree ($\alpha$), we propose a time-constraint IRNN query processing algorithm (TC-IRNN). Figure 2 shows our TC-IRNN query processing algorithm. At first, our algorithm calculates a distance limit by using the original time constraint and $\alpha$, and it finds candidate nodes located within the distance limit (lines 2–3). Next, the algorithm finds a set of candidate POIs from the candidate nodes, where a candidate POI is the nearest neighbor of its corresponding node (line 4–5). Finally, it finds the

```
TC-IRNN (BaseRoute, time_boundary, α)
// BaseRoute: query route, time_boundary : time constraint,
  α: weight for time constraint
1. CostTSRoute = infinite;
2. Boundary = (speed * time_boundary)/α;
3  Cand_nodes = Find_node(givenroute, Boundary);
4. for (i = 0; i > sizeof(cand_nodes) ;i++)
5. Cand_Set = Calculate_cost(Find_NN(Node(i), Distance));
6. return find_lowcost(Cand_Set);
End TC-IRNN Algorithm
```

**Fig. 2** TC-IRNN query processing algorithm.

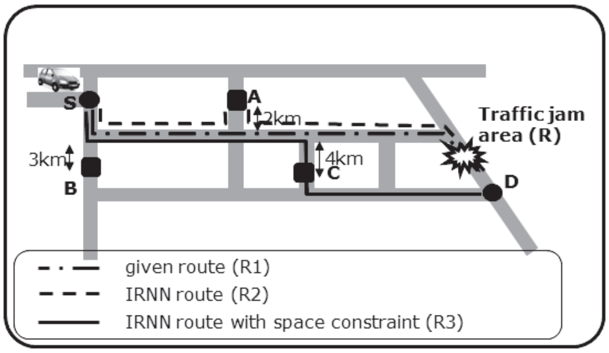

**Fig. 3** IRNN Query with space constraint.

best POI with the minimum deviation among a set of candidate POIs (line 6).

## 3.2 Space-Constraint IRNN Query Processing Algorithm

If there is road obstruction due to accident or traffic jam while a moving vehicle moves on a planned route, it is necessary to find an alternated route. For instance, a query is to find the best route with minimum deviation from the planned route R1 when R1 has a traffic jam area R, as shown in Fig. 3. The existing IRNN algorithm focuses only on finding a nearest neighbor which has a minimum distance from the given route. Therefore, the existing IRNN query processing algorithm selects a route R2 passing through a POI A. However, because traveling the traffic jam area may cause high cost, it is necessary to design a IRNN query processing algorithm considering space constraint. In the case of a light traffic jam on a road, we usually travel the planned route even though there is traffic jam. But, in the case of a heavy traffic jam due to car accident, we may select a new route which does not pass through the accident area. For this, we define a deviated route as follows.

***Definition 2*** *When a planned route is from $N_1$ node to $N_p$ node and a traffic jam area is in a segment from $N_i$ to $N_j$, we define a deviated route as a route which does not pass through the traffic jam area while travelling from $N_1$ to $N_p$.*

In Fig 4, there is a deviated route being composed of three parts, i.e., $N_1$-$N_i$, $N_i$-POI$_2$-$N_i$ and $N_j$-$N_p$. Among them, a deviated portion is $N_i$-$POI_2$-$N_i$ (dashed line) and the traveled portion of the planned route is $N_1$-$N_i$ and $N_j$-$N_p$.

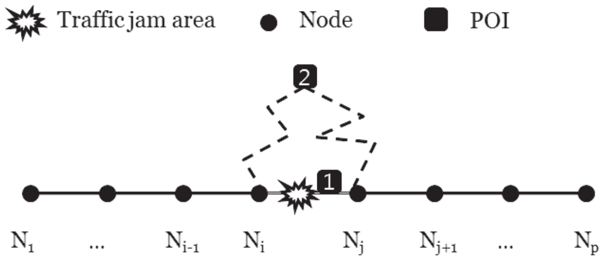

**Fig. 4** An example of cost function to avoid traffic jam area.

```
Compute_NodetoNN Algorithm (POIs, Network)
// POIs : set of POIs, Network = {Nodes, Edges}
1. POI poi, Nodes = Select_Nodes(Network);
2. poi = Random_Select_POI(POIs);
3  Node = Extend_POI(poi);
4. while (Stack is NULL) {
5.      Extend_Node(Node);
6.      Push(adjacency node(Node));
7.      temp_memory = Compute_info(adjacency node(Node));
8.      Stack = pop_stack(); }
End Compute_NodetoNN Algorithm
```

**Fig. 5** Pre-processing for storing NN and its distance from each node.

The space constraint portion means the segment(s) of the traffic jam area, i.e., $N_i$-$N_j$. The deviated portion provides a penalty because the exiting IRNN query processing algorithm tends to keep the given planned route. Based on the above three portions made from the deviated route, we can define a cost function for traveling a deviated route or a given panned route as follows.

***Definition 3*** *Assume that $|DP|$, $|TPR|$, and $|SCP|$ represent the length of the deviated portion, that of the traveled portion of the planned route, and that of the space constraint portion, respectively. We define the cost function of traveling a deviated route or the planned route where $\beta$ is the weight of user preference to the planned route, $\gamma$ is a penalty degree of not passing through the planned route, and $\delta$ is a burden of passing through a space constraint portion.*

$$Cost\_Route = \begin{cases} (1-\beta)*|DP| + \delta*|SCP|, & \text{in case of planned route} \\ (1-\beta)*|DP| + \beta*|TPR| + \gamma*|SCP|, & \\ & \text{in case of deviated route} \end{cases}$$

To support the efficient processing of our IRNN algorithm considering space constraint, we provide a preprocessing algorithm for finding the nearest neighbor (NN) of each node as shown in Fig. 5. At first, we select a POI and store two nodes of a segment including the POI into a stack which is used for network expansion. We named the stack as Stack_NE. For each node, we also store its nearest neighbor POI and the distance from the POI into another stack, Stack_NN. Secondly, from Stack_NE, we pop a node having shorter distance to the POI. Thirdly, we start network expansion from the popped node. If we find a POI which is nearer from a node than the previous POI, we change the nearest POI and its distance to the node in both Stack_NE

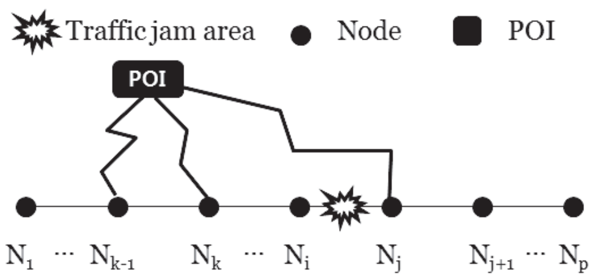

**Fig. 6** Example of two nodes containing same NN POI.

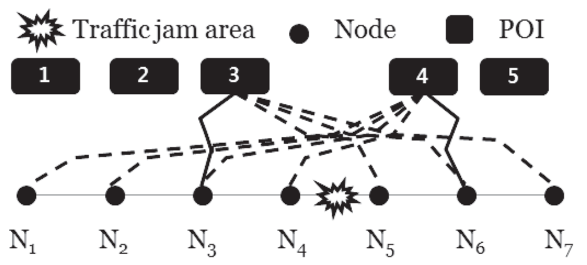

**Fig. 7** The number of possible deviated routes for a planned route.

and Stack_NN. Finally, we continues until the Stack_NE becomes empty. Since the network is expanded just a single time, our algorithm is efficient to find the nearest POI from each node.

To avoid unnecessary computation while searching a deviated path, we present theorem 1 and 2 for two routes having the same POI as nearest neighbor.

***Theorem 1*** *Assume that a given planned route is from $N_1$ node to $N_p$ node and a traffic jam area is in a segment between $N_i$ and $N_j$. If Distance$(N_{k-1}, POI)$ > Distance$(N_k, POI)$, the route being composed of $N_1$-$N_{k-1}$, $N_{k-1}$-POI-$N_j$, and $N_j$-$N_p$ can be discarded by the route composed of $N_1$-$N_k$, $N_k$-POI-$N_j$, and $N_j$-$N_p$, where $0 < k \leq i$, k is integer.*

*Proof: Assuming the penalty degree $\gamma \geq 1$ and $1 \geq \beta \geq 0$, according to definition 3, the cost of the former route is $(1-\beta) * ((Distance(N_1, N_{k-1}) + Distance(N_j, N_p)) + \beta * ((Distance(N_{k-1}, POI) + Distance(POI, N_j)) + \gamma * Distance(N_{k-1}, N_j)$, whereas the cost of the latter route is $(1-\beta) * ((Distance(N_1, N_k) + Distance(N_j, N_p)) + \beta * ((Distance(N_k, POI) + Distance(POI, N_j)) + \gamma * Distance(N_k, N_j)$. By subtracting the cost of the latter route from that of former route, we can obtain $(\gamma - 1 + \beta) * ((Distance(N_{k-1}, N_k) + \beta * ((Distance(N_{k-1}, POI) - (Distance(N_k, POI))$. This is always greater than zero because $Distance(N_{k-1}, N_k)$, $(\gamma - 1 + \beta)$, and $(Distance(N_{k-1}, POI) - (Distance(N_k, POI)$ have a positive value. Because the cost of the former route is greater than that of the latter route, the former can be discarded by the latter.*

***Theorem 2*** *Assume that a given planned route is from $N_1$ node to $N_p$ node and a traffic jam area in a segment between $N_i$ and $N_j$. If Distance$(N_{m+1}, POI)$ > Distance$(N_m, POI)$, the route being composed of $N_1$-$N_i$, $N_i$-POI-$N_{m+1}$, and $N_{m+1}$-$N_p$ can be discarded by the route composed of $N_1$-$N_i$, $N_i$-POI-$N_m$, and $N_m$-$N_p$, where $i < m \leq p$, m is integer.*

*Proof: This theorem can be proved in the same way as the theorem 1.*

Figure 6 shows an example for finding an optimal deviated route where two nodes ($N_{k-1}$ and $N_k$) having the same POI are located before the traffic jam area. Here, there are two possible deviated routes, such as one being composed of $N_1$-$N_{k-1}$, $N_{k-1}$-POI-$N_j$, and $N_j$-$N_p$ (i.e., $N_1$-$N_{k-1}$-POI-$N_j$-$N_p$) and one being composed of $N_1$-$N_k$, $N_k$-POI-$N_j$, and $N_j$-$N_p$ (i.e., $N_1$-$N_k$-POI-$N_j$-$N_p$). If the distance between $N_{k-1}$ and POI is greater than the distance between $N_k$ and POI, the deviated portion ($N_{k-1}$-POI-$N_j$) in the former route is greater than the deviated portion ($N_k$-POI-$N_j$) in the latter route. Therefore, we can discard the former route (i.e., $N_1$-$N_{k-1}$-POI-$N_j$-$N_p$) because it has a greater distance.

***Definition 4*** *Assume that a given planned route is from $N_1$ node to $N_p$ node and a traffic jam area is in a segment between $N_i$ and $N_j$. If X and Y are the total number of NN POIs for the nodes located before and after the traffic jam area, respectively, the number of possible deviated routes can be computed as follows.*

$$\begin{aligned}&\textit{Number of possible deviated routes}\\&= X * (p - m) + Y * m,\end{aligned}$$

*where m is the number of nodes before the traffic jam area.*

For example, a planned route has 7 nodes (p = 7) and the number of nodes before the traffic jam area is m = 4, as shown in Fig. 7. The total number of NN POIs for seven nodes is 5. Among them, the number of NN POIs for nodes located before the traffic jam area is X = 3 and that for nodes located after the traffic jam area is Y = 2. By using definition 4, the total number of possible deviated routes is $3 * 3 + 2 * 4 = 17$.

In order to reduce the cost of network expansion, we select a NN POI with the shortest distance, out of K NN POIs for the nodes. For this, we provide two definitions for ordering K NN POIs.

***Definition 5*** *Assume that a given planned route is from $N_1$ node to $N_p$ node and a traffic jam area is in a segment between $N_i$ and $N_j$. If two nodes $N_b$ and $N_c$ being located before the traffic jam area have $POI_B$ and $POI_C$ as NN POI, respectively, the order of network expansion is given below. If $\beta * Distance(N_b, N_c) + \gamma * Distance(N_b, POI_B) > \beta * Distance(N_c, POI_C)) + (1-\beta) * Distance(N_b, N_c)$, the network expansion starts from $POI_B$. Otherwise, the network expansion starts from $POI_C$.*

***Definition 6*** *Assume that a given planned route is from $N_1$ node to $N_p$ node and a traffic jam area is in a segment between $N_i$ and $N_j$. If two nodes $N_b$ and $N_c$ being located after the traffic jam area have $POI_B$ and $POI_C$ as NN POI, respectively, the order of network expansion is given below. If $\beta * Distance(N_b, N_c) + \gamma * Distance(N_b, POI_B) > \beta * Distance(N_c, POI_C)) + (1-\beta) * Distance(N_b, N_c)$, network expansion starts from $POI_B$. Otherwise, the network expansion starts from $POI_C$.*

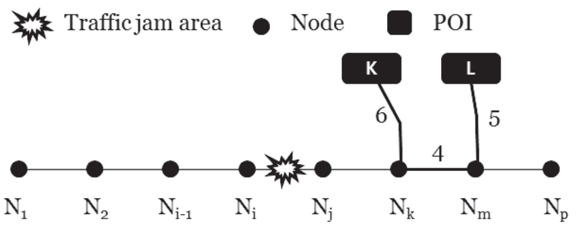

**Fig. 8** Ordering of network expansion between two nodes.

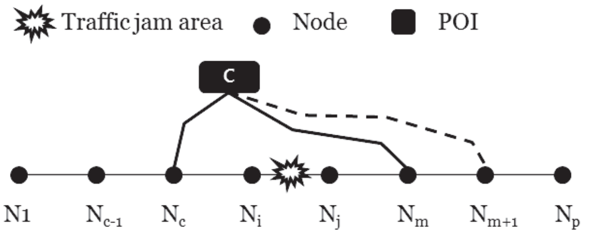

**Fig. 9** Pruning method for network expansion.

Figure 8 shows an ordering of network expansion for two nodes $N_k$ and $N_m$ ($k < m$) being located after the traffic jam area. When $\beta$ is 0.8 and $\gamma$ is 1, we start the network expansion from the $POI_K$ first because the sum of the cost of $N_k$-$POI_K$ and the additional TPR cost ($= 0.8*6+0.2*4 = 4.2$) is less than the sum of the cost of $N_m$-$POI_M$ and the additional SCP cost ($= 0.8 * 5 + 1 * 4 = 8$).

Once we select NN POIs before and after the traffic jam area, we perform the network expansion from both sides, based on the order of NN POIs which has been computed. Because we cannot decide the order between NN POIs which have come from both sides, we select one by one from both sides, alternatively. After deciding the order of NN POIs, we provide a pruning method to reduce the network expansion further.

***Definition 7*** *Assume that a given planned route is from $N_1$ node to $N_p$ node, a traffic jam area is in a segment between $N_i$ and $N_j$, and a nodes $N_c$ being located before the traffic jam area have $POI_C$ as NN POI. If $N_m$ is shortest from $POI_C$ among nodes located after the traffic jam area, we can discard the computation of nodes which are located far from $N_m$.*

For example, when $POI_C$ is the NN POI of $N_c$in Fig. 9, the distance between $POI_C$ and $N_m$ is the shortest among those between $POI_C$ and other nodes being located in the opposite side of $POI_C$. Therefore, we can reduce the network expansion by discarding deviated routes including $POI_C$-$N_{m+1}$ to $POI_C$-$N_p$.

Based on the above definitions 2–7, we propose a space-constraint IRNN query processing algorithm (SC-IRNN). Figure 10 shows our SC-IRNN query processing algorithm. At first, our algorithm divides a planned route into two parts: before a space constraint area and after a space constraint area (lines 3–4). Next, it retrieves the candidate set of POIs from each node for both parts (lines 5–6). Then, it searches the shortest deviated route by using the retrieved candidate set of POIs (lines 7–16). During this process, it reduces the network expansion by setting an upper bound as a travel cost passing through the space constraint area, thus leading to the reduction of both computation and I/O cost.

```
SC-IRNN Algorithm(BaseRoute, Link, β, δ, γ)
// BaseRoute: query route, Link : traffic jam area,
  β : weight for space constraint, δ : weight for traffic jam area,
  γ : penalty for detouring route
 1.  CostSCRoute = infinite;
 2.  Low_Cost = infinite; temp_POI = 0;
 3.  LeftRoute = DivideRoute_Left(BaseRoute, Link);
 4.  RightRoute = DivideRoute_Right(BaseRoute, Link);
 5.  for (i = 0; i > sizeof(LeftRoute) ;i++)
       CandidateSet(Before) = Find_CandidateSet(Node(i),
       Distance, β, δ, γ);
 6.  for (i = sizeof(LeftRoute)+1; i > sizeof(TotalRoute); i++)
       CandidateSet(After)   =   Find_CandidateSet(Node(i),
       Distance, β, δ, γ);
 7.  UpperBound =
         Calculate_Cost(Route_PassingConstraint(Link), β, δ);
 8.  while (CandidateSet(Before) != {∅} &&
                CandidateSet(After) != {∅}) {
 9.      temp_Node = select_node(CandidateSet(Before));
10.      CandidateSet(Before) –= {temp_Node};
11.      temp_Cost = Calculate_Cost(temp_Node, β, δ, γ);
12.      if (temp_Cost < UpperBound) {
                UpperBound = temp_Cost; }
13.      temp_Node = select_node(CandidateSet(After));
14.      CandidateSet(Before) –= {temp_Node}
15.      temp_Cost = Calculate_Cost(temp_Node, β, δ, γ);
16.      if (temp_Cost < UpperBound) {
            UpperBound = temp_Cost; } }
End SC-IRNN Algorithm
```

**Fig. 10** SC-IRNN query processing algorithm.

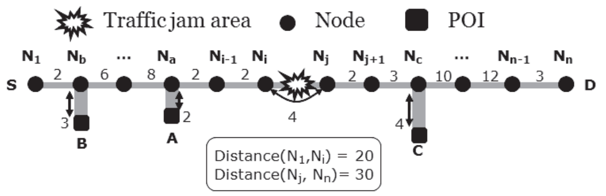

**Fig. 11** Example of IRNN query processing algorithm with space constraint.

Figure 11 shows an example of the SC-IRNN query processing algorithm. Assume that a user (car) plans to travel from node $N_1$ ($S$) to node $N_n$ ($D$) and $\beta$, $\delta$, $\gamma$ are 0.8, 10, 2, respectively. We also assume that the distance from $N_1$ to $N_i$ is 20 and the distance from $N_j$ to $N_n$ is 30, and the length of space constraint area is 4. In this example, the cost of a route passing through the space constraint area, i.e., UpperBound, is computed below.

$$\begin{aligned}\text{UpperBound} &= (1 - 0.8) * (20 + 4 + 30) + (10 * 4)\\ &= 50.8.\end{aligned}$$

There are three POIs in the planned route, i.e., A, B, and C. Such nodes as $N_a$, $N_{i-1}$, and $N_i$ have the same $\text{POI}_\text{A}$

as NN POI. Among them, the closest node from the $\mathrm{POI_A}$ is $N_a$. By using definition 3, we can see that the closest nodes of three POIs are $N_a$, $N_b$, and $N_c$, respectively. Next, we calculate the costs of the deviated routes including the POIs A, B, and C, respectively. Finally, we can select the best deviated route with the minimum cost. If the cost of deviated route is greater than UpperBound, our algorithm stops calculation.

## 3.3 Time- and Space-Constraint IRNN Query Processing Algorithm

Based on our previous IRNN query processing algorithms, we propose a time- and space-constraint IRNN query processing algorithm (TSC-IRNN). An example of TSC-IRNN query is shown like this: "For a given time constraint and a given traffic jam area (space constraint), find a route which is deviated minimally from the traffic jam area while passing through a POI being located within the time constraint". For processing this query, there are three solutions: i) considering time and space constraint at the same time, ii) considering space constraint after time constraint, and iii) considering time constraint after space constraint. Among them, the solution to consider time constraint after space constraint has the worst performance because the processing time for space constraint is always expensive. The first and second solutions can achieve better performance because they can reduce the length of a planned route to be processed by using time constraint. Therefore, we consider only the first and second solutions for designing our TSC-IRNN query processing algorithm.

To deal with a TSC-IRNN query in an efficient way, we decide which solution is appropriate for given time and space constraints as follows.

***Definition 8*** *Assume that a planned route is from $N_1$ node to $N_p$ node and the distance from $N_1$ to the space constraint area is $S$. If the distance limit of time constraint is greater than and equal to $S$, we select a query processing approach which consider time and space constraint at the same time. Otherwise, we select an approach to consider space constraint after time constraint.*

To deal with the first solution, we set an UpperBound as the cost of travelling a route which passes through a given space constraint area. Secondly, the portion of planned route within the distance limit of time constraint (DL) is divided into two parts: before space constraint and after space constraint. Thirdly, the algorithm finds the best deviated route with the minimum cost among all possible deviated routes. The first solution can reduce both computation and I/O costs because it can process only a portion of the planned route by using the UpperBound. If the number of nodes located before the space constraint area and that located within DL are $I$ and $N$, respectively, the computation cost of the first solution is $I * (N - I)$.

Figure 12 shows an example of space constraint area located within DL. Here, the planned route is represented as dashed line, the original IRNN route is represented as

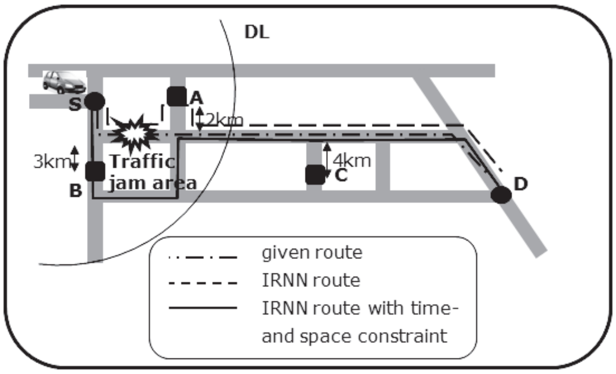

**Fig. 12** Space constraint area located within DL.

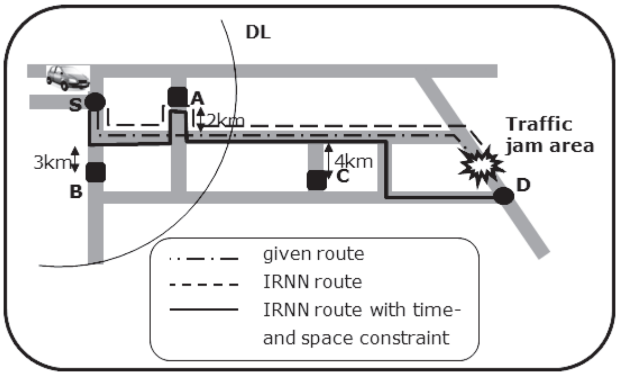

**Fig. 13** Space constraint area located beyond DL.

chain line, and DL is shown as a curve. We see that POIs A and B are located within DL. To find the best deviated route from the planned route, we can compute only a portion of the planned route, i.e., from S to DL, By using our algorithm, we can find the best deviated route (shown as continuous line) that does not pass through the space constraint area, but drops by POI B. Because our algorithm does not need to process the remaining part of the planned route, i.e., from DL to D, it is efficient in terms of I/O and computation costs.

For the second solution, our algorithm has to process the time constraint before handling the space constraint. To search an appropriate POI, our algorithm uses pre-computed NN for each node within DL. Because our algorithm does not need to search POIs located beyond DL, it can reduce both computation and I/O costs. If the number of nodes located before and after space constraint area is $I$ and $M$ respectively, and the number of nodes located within DL is $N$, the computation cost of the second solution is $(I - N) * M$.

Figure 13 shows an example of space constraint area located beyond DL. The $\mathrm{POI_A}$ and $\mathrm{POI_B}$ are located within DL whereas $\mathrm{POI_C}$ lies beyond DL. To find the nearer POI between $\mathrm{POI_A}$ and $\mathrm{POI_B}$, we compare the distances of the two POIs from their respective nodes. Next, we find the best deviated route with the minimum cost by using UpperBound. Finally, we return the best deviated route as one being represented as continuous line.

```
TSC-IRNN (BaseRoute, time_boundary,Link, $\alpha$, $\beta$, $\delta$, $\gamma$)
// BaseRoute: query route, time_boundary : time constraint,
 Link : traffic jam area, $\alpha$: weight for time constraint,
$\beta$ : weight for space constraint, $\delta$ : weight for traffic jam area,
$\gamma$ : penalty for detouring route
1. CostTSCRoute = infinite; Low_Cost = infinite;
temp_POI = 0;
2. Boundary = (speed * time_boundary)/$\alpha$;
3. If (Boundary <= Distance (StartNode, Link)); {
4.   LeftRoute = DivideRoute_Left(BaseRoute, Link);
5.   RightRoute = DivideRoute_Right(BaseRoute, Link); }
6. else {
7.    LeftRoute = DivideRoute_Left(BaseRoute, Link);
8.    RightRoute = DivideRoute_Right(BaseRoute, Link); }
9. Upp_Bound =
Calculate_Cost(Route_PassingConstraint(Link),
      $\beta$, $\delta$);
10. for (i = 0; i > sizeof(LeftRoute) ;i++)
11.   CandidateSet(Before) =
Find_Cand_Set(Node(i), Distance, $\beta$, $\delta$, $\gamma$);
12. for (i = sizeof(LeftRoute)+1; i > sizeof(TotalRoute) ;i++)
13.   CandidateSet(After) =
Find_Cand_Set(Node(i), Distance, $\beta$, $\delta$, $\gamma$);
14. while (CandidateSet(Before) != {$\varnothing$}) {
15.    NodeB = select_node(CandidateSet(Before));
16.    while (each Node for CandidateSet(After)) {
17.       temp_Cost = Calculate_Cost(NodeB, Node, $\beta$, $\delta$, $\gamma$);
18.       if (temp_Cost < Upp_Bound) {
   Upp_Bound = temp_Cost; } }
End TSCIRNN Algorithm
```

**Fig. 14** TSC-IRNN query processing algorithm.

Figure 14 shows our TSC-IRNN algorithm which is based on Definitions 2~6. At first, according to the position of space constraint area, this algorithm divides a planned route into two parts: before space constraint and after space constraint (lines 2–8). In order to reduce both computation and I/O costs, our TSC-IRNN algorithm sets as UpperBound a cost for travelling a route passing through the space constraint area (line 9). Next, it finds a candidate POI set from each node of the planned route (lines 10–13). Finally, it returns a route with the lowest cost, i.e., the best route passing through the nearest POI among the POIs of the two candidate sets (lines 14–18).

## 4. Performance Analysis

To show the efficiency of our three algorithms, we compare them with the PCZ algorithm which is the best one among the existing IRNN algorithms. We do our experiments by using the San Francisco Bay road network data [11]. The San Francisco Bay map data consists of approximately 220,000 edges and 170,000 nodes. By using RunTime21 algorithm [12], we randomly generate 10,846 POIs with different types, such as gas stations, restaurants and so on, as shown in Fig. 15. We decide time constraint by using the length of the given route and the speed of a moving object. Because we assume that the speed of the moving object is constant, time constraint depends on the length of the given route. In addition, we decide space constraint by selecting one edge randomly from the given route.

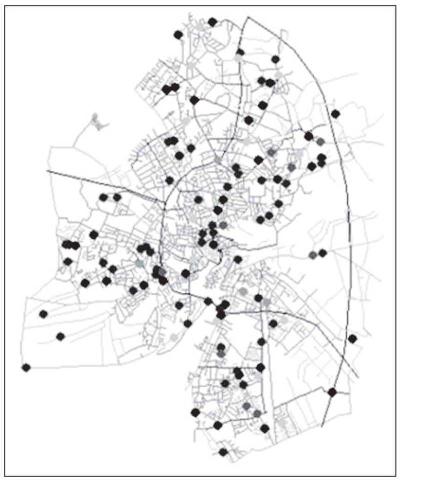

**Fig. 15** San Francisco Bay road network with POIs.

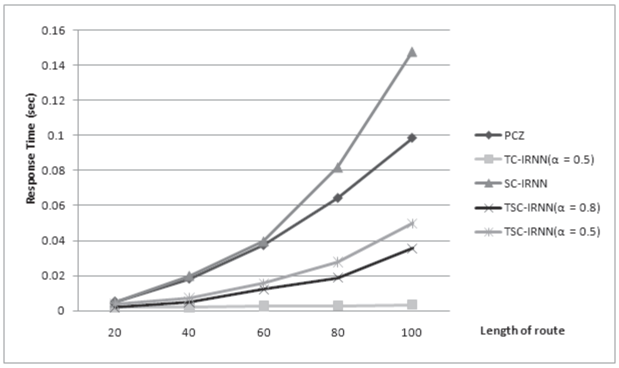

**Fig. 16** Total Response time ($\beta$: 0.8, $\gamma$: 1.0, $\delta$: 2.0).

Figure 16 show the performance comparison of our three IRNN query processing algorithms with the existing PCZ algorithm, for routes consisting of 20, 40, 60, 80, and 100 nodes, when $\beta = 0.8$, $\gamma = 1.0$, and $\delta = 2.0$. In case of a route being composed of 60 nodes, the response times of TC-IRNN, TSC-IRNN ($\alpha$ = 0.2), TSC-IRNN ($\alpha$ = 0.5), SC-IRNN, PCZ algorithms are 0.003, 0.005, 0.015, 0.040, and 0.037 seconds, respectively. It is shown that our TC-IRNN algorithm has the best performance because it needs to search POIs only within the distance limit of time constraint. Whereas, our SC-IRNN algorithm achieves the nearly same performance as the PCZ algorithm because it has the overheads of finding the candidate set of deviated routes even though it uses efficient pruning techniques. In addition, our TSC-IRNN algorithm has better performance than the existing PCZ algorithm even though we consider both time and space constraint. This is because we can reduce the overhead of finding a deviated route by using both the distance limit of time constraint and efficient pruning techniques. Among TSC-IRNN algorithms, our TSC-IRNN algorithms with $\alpha = 0.2$ is better than that with $\alpha = 0.5$ because only the short portion of a route can be processed when $\alpha$ is small. In case of a route being composed of 100 nodes, the response times of TC-IRNN, TSC-IRNN

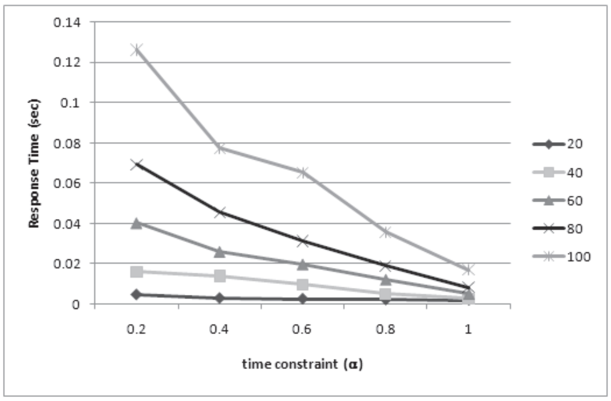

**Fig. 17** Response time of TSC-IRNN algorithm in term of $\alpha$.

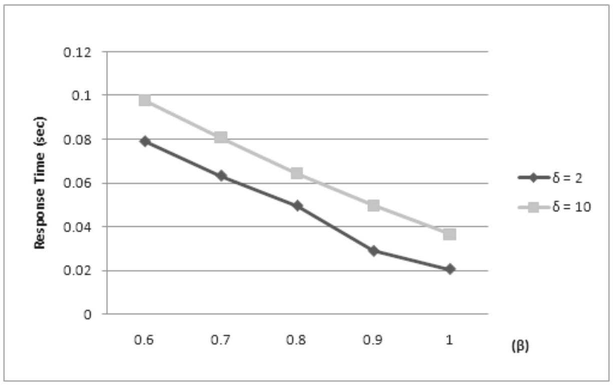

**Fig. 18** Response time of TSC-IRNN algorithm in term of $\beta$.

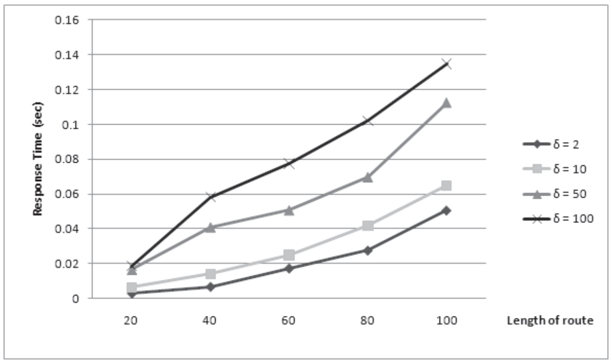

**Fig. 19** Response time of TSC-IRNN algorithm in term of $\delta$.

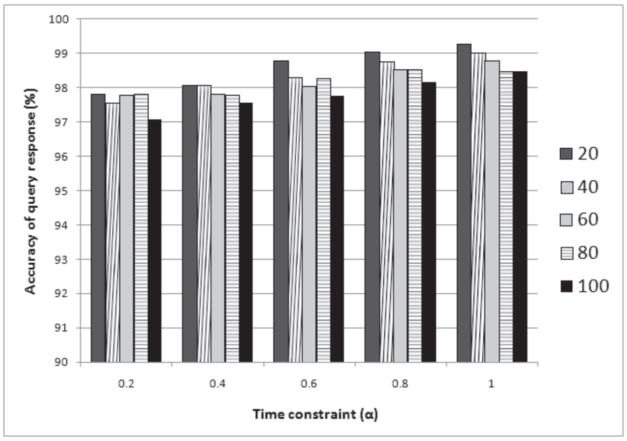

**Fig. 20** Accuracy of query response as compared to the optimal route.

($\alpha$ = 0.2), TSC-IRNN ($\alpha$ = 0.5), SC-IRNN, PCZ algorithms are 0.004, 0.016, 0.049, 0.148, and 0.098 seconds, respectively. The performances patterns of our algorithms are the nearly same as those with a route being composed of 60 nodes. However, it is shown that our SC-IRNN algorithm has the worst performance because the overheads of finding the candidate set of long deviated routes overwhelm the benefit of using its efficient pruning techniques.

To measure the performances of our three IRNN query processing algorithms in terms of main designing parameters, we do our experiment on our TSC-IRNN query processing algorithm because it can cover both TC-IRNN and SC-IRNN algorithms. Figure 17 shows the response time of our TSC-IRNN algorithm in term of $\alpha$, for routes consisting of 20, 40, 60, 80, and 100 nodes, when $\beta = 0.8$, $\gamma = 1.0$, and $\delta = 2.0$. This shows the impact of the different values of $\alpha$, i.e., the tightness degree of time constraint. For $\alpha = 0.8$, the response times are 0.002, 0.005, 0.012, 0.019, and 0.036 seconds when the lengths of planned routes are 20, 40, 60, 80, and 100, respectively. In addition, it is shown that the response time decreases according to the increase of $\alpha$. The reason is because DL decreases as the value of $\alpha$ increases. As a result, a response time can be reduced when DL is short because DL contains the small number of nodes to be processed.

Figure 18 shows the response time of our TSC-IRNN algorithm in term of $\beta$, when $\alpha = 0.5$, $\gamma = 1.0$, and the route consists of 100 nodes. This shows the impact of the different values of $\beta$, i.e., a user's preference to follow the given query route. When the value of $\beta$ is 0.8 and the value $\delta$ is 2.0, our TSC-IRNN algorithm requires 0.049 seconds. When the value of $\delta$ is 10, it requires 0.064 seconds. It is shown that the retrieval time decreases as the values of $\beta$ increases. This is because our pruning techniques can reduce the number of deviated routes to be processed.

Figure 19 shows the retrieval time of our TSC-IRNN algorithm in terms of $\delta$, when $\alpha = 0.5, \beta = 0.8$, and $\gamma = 1.0$. This shows the impact of the different values of $\delta$, i.e., the penalty of the traffic jam area within the given query route. When the penalty values of $\delta$ are 2, 10, 50, and 100, the response times are 0.050, 0.064, 0.112, and 0.134 seconds respectively, where the length of planned route is 100. It is shown that the response time is increased as the length of route increases as well as $\delta$ increases. This is because our algorithm has to process the larger number of nodes from the deviated routes with the increase in both $\delta$ value and route length.

Figure 20 shows how good the query response is. For this, we compare the optimal route with the computed route which is acquired based on the cost function of a detour route in Definition 3. The optimal route is one having the minimum cost among all possible routes between a start node and a destination node. The result shows that the accuracy of query response (our computed routes) is about 98.2% on the average as compared with the optimal ones. This is because our algorithm generally selects the shortest detour routes visiting NN POIs which are the optimal routes. But, our algorithm does not select the optimal routes in few cases since there may exist the optimal routes which do not pass through NN POIs. As shown in Fig. 20, the probability of the cases is only about 2%, so it is shown that our algorithm can find the optimal routes in almost all cases.

## 5. Conclusions and Future Work

Since moving objects move on predefined spatial networks, route-based queries are essential in the spatial network databases. As a typical route-based query, the in-route nearest neighbor (IRNN) query [6] was proposed to focus on finding a nearest neighbor with minimal deviation from a given route. But, the existing IRNN query processing algorithm has a problem that it cannot consider real-time situation on road networks, such as a car with limited oil or a traffic jam situation. To overcome this problem, we proposed three query processing algorithms considering both time and space constraints, i.e., TC-IRNN, SC-IRNN, and TSC-IRNN. Our IRNN query processing algorithms find the best deviated route with the minimum cost in real applications, such as Telematics, CNS and automatic navigation devices. Through our performance analysis, it is shown that the response time of our TC-IRNN and TSC-IRNN query processing algorithms are better than the existing IRNN query processing algorithm. As future work, we need to apply our IRNN query processing algorithms considering time and space constraints to real road network applications so that we can prove their effectiveness.

## Acknowledgments

This research was financially supported by the Ministry of Education, Science Technology (MEST) and Korea Institute for Advancement of Technology (KIAT) through the Human Resource Training Project for Regional Innovation. This research was also supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency)" (NIPA-2010-(C1090-1021-0009))

### References

[1] Z. Song, and N. Roussopoulos, "K-nearest neighbor search for moving query point," Proc. SSTD, pp.79–96, 2001.

[2] Y. Tao and D. Papadias, "Time parameterized queries in spatio-temporal databases," Proc. ACM SIGMOD, pp.334–345, 2002.

[3] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," Proc. VLDB, pp.287–298, 2002.

[4] M.R. Kolahdouzan and C. Shahabi, "Continuous k nearest neighbor queries in spatial network databases," Proc. STDBM, pp.33–40, 2004.

[5] K. Mouratidis, M.L. Yiu, D. Papadias, and N. Mamoulis, "Continuous nearest neighbor monitoring in road networks," Proc. VLDB, pp.43–54, 2006.

[6] S. Shekhar and J.S. Yoo, "Processing in-route nearest neighbor queries: A comparison of alternative approaches," Proc. ACM GIS, pp.9–16, 2003.

[7] Y. Hsueh, R. Zimmermann, and M. Yang, "Approximate continuous K nearest neighbor queries for continuous moving objects with pre-defined paths," Proc. COMOGIS, pp.270–279, 2005.

[8] H. Jung, S. Kang, M. Song, S. Im, J. Kim, and C. Hwang, "Towards real-time processing of monitoring continuous k-nearest neighbor queries," Proc. ISPA, pp.11–20, 2006.

[9] J. Feng, L. Wu, Y. Zhu, N. Mukai, and T. Watanabe, "Continuous k-nearest neighbor search under mobile environment," Proc. WAIM, pp.566–573, 2007.

[10] E.W. Dijkstra, "A note on two problems in connection with graphs," Numeriche Mathematik, vol.1, pp.269–271, 1959.

[11] T. Brinkhoff, "A framework for generating network — Based moving objects," GeoInformatica, pp.153–180, 2002.

[12] http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/

**Yong-Ki Kim** is a Ph. D. in the Chonbuk National University. He received the B.S., M.S. and Ph. D. degrees in Chonbuk National University in 2002, 2005, and 2011, respectively. His research interests include spatial network database, query processing and sensor network.

**Jae-Woo Chang** is a professor in the Department of Computer Engineering, Chonbuk National University, Korea from 1991. He received the B.S. degrees in Computer Engineering from Seoul National University in 1984. He received the M.S. and Ph. D degrees in Computer Engineering from Korea Advanced Institute of Science and Technology (KAIST) in 1986 and 1991, respectively. During 1996–1997, he stayed in University of Minnesota for visiting scholar. And 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.