An Edge-Based Algorithm for Spatial Query Processing in Real-Life Road Networks

Ye-In Chang, Meng-Hsuan Tsai, and Xu-Lun Wu

***Abstract*—Due to wireless communication technologies, positioning technologies, and mobile computing develop quickly, mobile services are becoming practical and important on big spatiotemporal databases management. Mobile service users move only inside a spatial network, e.g. a road network. They often issue the *K* Nearest Neighbor (*K*NN) query to obtain data objects reachable through the road network. The challenge problem of mobile services is how to efficiently answer the data objects which user interest to the corresponding mobile users. Lu *et al*. have proposed a RNG (Road Network Grid) index for speeding up the *K*NN query on real-life road networks. Since they divide the road, this makes the number of points of the graph increase. It increases the execution time of constructing the index structure. Therefore, in this paper, we propose a network model that captures the real-life road networks. We map the real-life road networks into graph directly. Then, based on our network model, we propose an EBNA (Edge-Based Nine-Area tree) index structure to make the search time of obtaining the interest edge information quickly. From our simulation result, we show that the performance of constructing the EBNA index is better than constructing the RNG index and the performance of the *K*NN query processing by using EBNA index is better than the *K*NN query processing by using RNG index.**

***Index Terms*—Edge-based, index structure, *K*NN, road network.**

I. Introduction

A spatial database system is a database system that contains a set of objects in space. It offers spatial data types in its data model and query language. Most existing researches in spatial database only consider the Euclidean space. The distance between two objects is determined by their relative position in space. However, in practice, the users can only move inside pre-defined road networks (road, railway, river, *etc*.). The distance between two objects in spatial network databases is the shortest path length of two objects rather than their Euclidean distance. Consequently, previous researches are not suitable in spatial network databases (SNDB) [1][2]. Many algorithms for answering variety spatial queries are developed. Spatial queries, or query arise over spatial data, can be classified into three main types: *spatial range queries*, *spatial join queries*, and *nearest neighbor queries*. The most one of nearest neighbor queries is K Nearest Neighbor query [3]. An example of *K* nearest neighbor query is "Find 5 restaurants that are the nearest to NSYSU." We usually answer the top *k* nearest neighbors to the user. The spatial network databases in real-life application become more and more important. Therefore, some studies force on spatial network database [4], [5]. The field of spatial network databases has to force on two main topics: 1) modeling the real-life road network and 2) indexing and query processing. The challenge of the spatial network database is how to transfer the real-life maps (networks) into a modeling graph. In this paper, we propose an Edge-Based Nine-Area tree (EBNA) index structure to solve the search time in the leaf node and speed up the *K*NN query. In the EBNA index structure, we do not have to partition the space to construct the index structure and we effectively obtain the edge information.

The rest of the paper is organized as follows. Section II gives a survey of some spatial queries in spatial network databases. Section III presents our proposed modeling, the partition number scheme use of in the EBNA index structure, and the *K*NN query processing. In Section IV, we give the performance of the proposed method. Finally, we give the conclusion in Section V.

II. Related Work

In the last decade, query processing in spatial databases develops quickly, consequently, a great number of query methods have been proposed. In this section, we give a survey of *K*NN query processing algorithms in spatial network databases. Papadias *et al*. proposed the Incremental Network Expansion (IER) method to compute *K*NN [6]. They proposed two approaches to answer the *K*NN queries on Euclidean distance and network distance. Yin *et al*. propose a *K*NN algorithm which uses a set of index structures to support the queries of moving object. Their method is that change the dynamic problem to the static problem.

Lu *et al*. proposes a road network model which captures the real-life road networks [7]. Based on the RNG index, the *K*NN and C*K*NN queries are proposed. The Road Network Grid (RNG) to partition the road network into cells, and generate a quad-tree to index all cells. 0-(a) shows an example of the grid partition, and 0-(b) shows the quad-tree which index all cells in 0-(a). In the quad-tree, each inner node records the corresponding cell positions and the pointer of four subcells. The leaf nodes record the corresponding cell positions and the information of all adjacent edges. 0 shows the leaf node format of RNG index. The *vl* records the positions of all vertices inside the cell, and the adjacent

Manuscript received April 28, 2015; revised July 18, 2015.

Y. I. Chang and M. H. Tsai are with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, 80424 Taiwan (e-mail: changyi@cse.nsysu.edu.tw, d023040002@student.nsysu.edu.tw).

X. L. Wu is with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, 80424 Taiwan. He is now with the Department of IT, Bread of Life Christian Church in Kaohsiung, 81365 Taiwan (e-mail: joseph_wu@bolkh.org.tw).

DOI: 10.7763/IJMO.2015.V5.480

vertex list *avl* records positions of all adjacent vertices. Each entry of *avl* represents an edge in the road network model. This entry has the length (*len*), the speed limitation (V_{sp}), and block factor (*bf*) of the edge. Each *avl* entry has a pointer that points to the object list *dp*, while the object list records positions of all the objects located at the corresponding edge. The flag *f* is used to distinguish between directed edges and undirected edges. They use the RNG index to process *K*NN queries. The *K*NN query algorithm requires the number of nearest neighbor *k*, the query object *q*, and the two end points (v_s and v_e) of the edge where the query is issued as inputs. The *avl* entry in the RNG index and the data objects along the edge can be obtained by using the star point v_s and the end point v_e. If the corresponding edge is a directed edge, the objects in the road from the query point to the end points can be obtained and the end point v_e is recorded for further exploring. Otherwise, it collects all the objects along the edge and records the two end points for further exploring. If the number of data objects is less than *k*, it has to explore other edges and the data objects will be obtained. The processing continues until the number of data objects not less than *k*, and the top *k* data objects is the *K*NNs (see Fig. 1-Fig. 2).

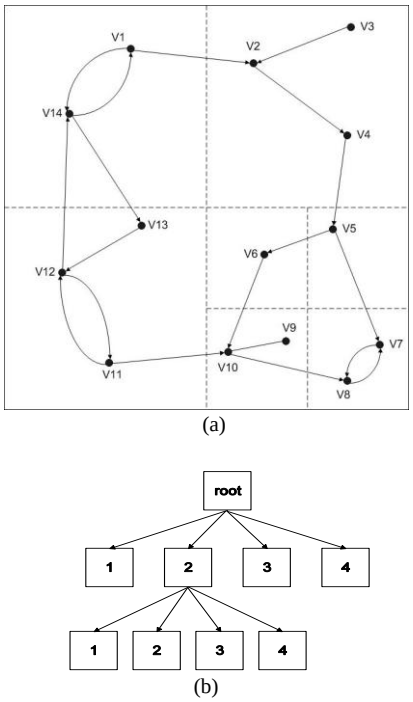


Fig. 1. The example of: (a) grid partition of road network; (b) a quad-tree index of grid partition.

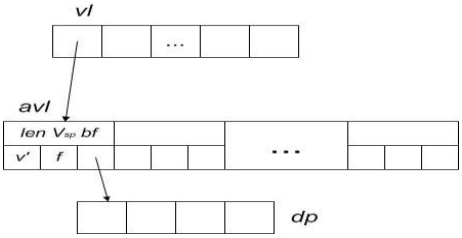


Fig. 2. Information of adjacent edges.

III. Edge-based Query Processing in Real-Life Road Networks

Due to the developments of mobile services, the query processing in the road (spatial) networks have become an important research field. The research works focus on three topics which are modeling the real-life network, efficiently indexing the road network, and query processing based on the index structure. Therefore, in this section, we describes the proposed road network model, indexing the road networks, and query processing based on our index structure.

A. The EBNA Index Structure

We propose the EBNA index that is an edge-based index structure. And we locate the spatial object on the spatial number, instead of the partitioning the spatial space. The road network consists of a number of edges. We store the details of edges on the NA-tree. We use the spatial number of two end points of the edge to decide which child belongs to. 0-(a) shows an example of the road network, and 0-(b) the corresponding NA-tree [8] (see Fig. 3).

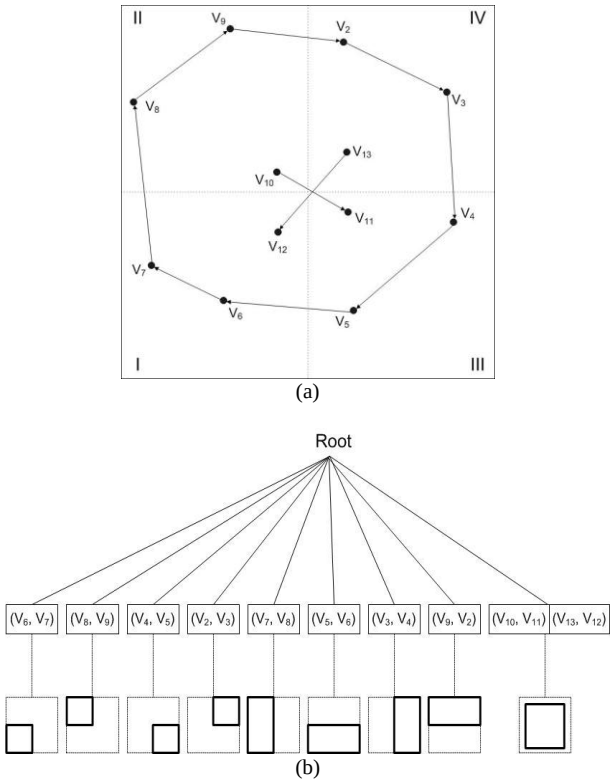


Fig. 3. An example: (a) the road network model; (b) the corresponding NA-tree.

In the NA-tree, each internal node stores the range of the spatial number and the pointer that points to children. Each leaf node stores the positions of two end points of an edge and the information that relates to the edge. Based on our road network model, we show how to store the data objects depending on the road type. 0-(a) shows an example that the road network with data objects and the corresponding road network model and tree structure are shown in 0-(b) and 0-(c). The V_1V_2 road is Type 2 road with U-turn, the leaf node stores the information of V_1 to V_2 edge and the information of V_2 to V_1. In this child, the *U_turn_pos* and *U_turn_Obj* store the position of U-turn that is a gray point in the 0-(b) and data objects locate at the U-turn position. The *ExpEdge* item has pointers to point to other leaf nodes, if the end point of the edge is the start point of other edges. Let's see an example of V_1 to V_2 edge in 0-(b). Since vertex V_2 is the end point of the V_1 to V_2 edge and is the start point of V_2 to V_1 edge, V_2 to V_4 edge, and V_2 to V_3 edge, the *ExpEdge* item of V_1 to V_2 edge

entry has three pointers to point to the V_2 to V_1 edge entry, the V_2 to V_4 edge entry, and the V_2 to V_3 edge entry. V_4V_5 and V_6V_4 is Type 1 road, it is mapped to only one directed edge. Therefore, the leaf node only has one entry. Since V_2V_4 is Type 3 road, it is mapped into two directed edges. The *ObjList* item of this two directed edge has same data objects. V_2V_3 is type 2 road that is mapped to two directed edge and the *ObjList* item stores the data objects along the side of the edge (see Fig. 4-Fig. 5).

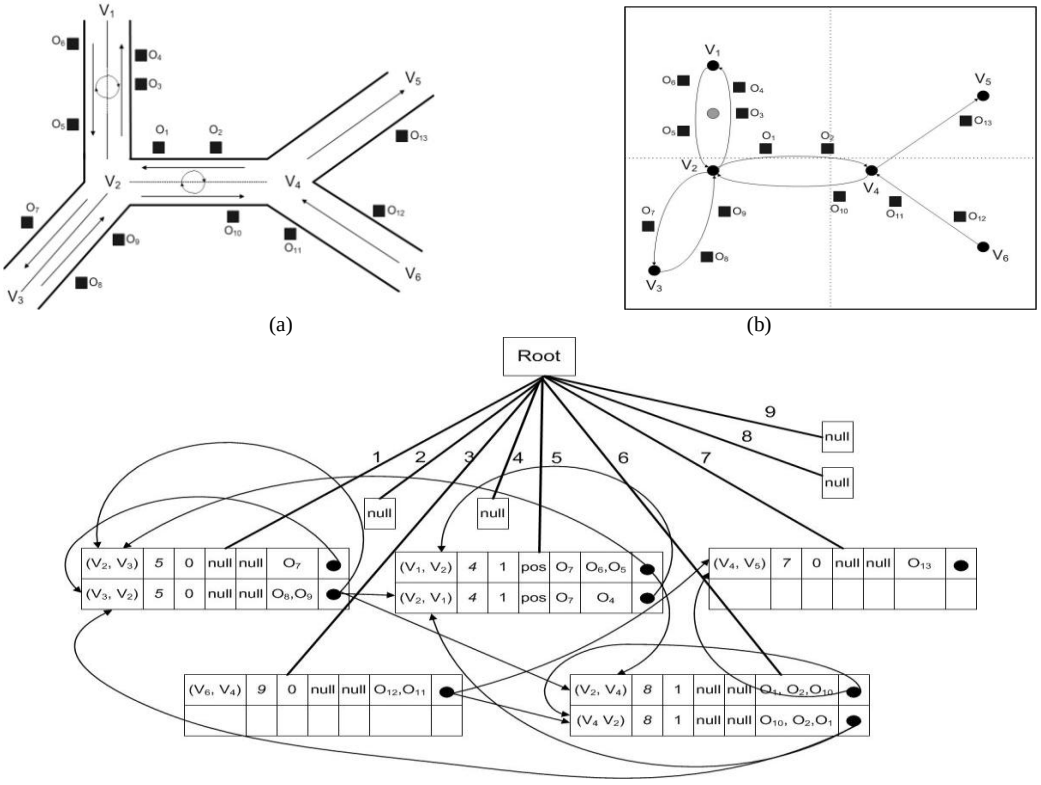


Fig. 4. The road network: (a) for example; (b) in the model; (c) by EBNA index structure.

```
1: Function Edge_Expansion(Edge_ExpQ, cur_KNN, K): K Nearest Neighbors;
2: Input:
3:         Edge_ExpQ: the queue of expansion edge
4:         cur_KNN: the current KNN
5:         K: the number of nearest neighbor
6: Output:
7:         Final_KNN: the array stores the KNN objects
6: begin
7:   if |cur_KNN| >= K then
8:     d_MAX := cur_KNN[K].distance
9:   else
10:    d_MAX := ∞;
11:  while (Edge_ExpQ ≠ ∅) do
12:  begin
13:    ExpE := Edge_ExpQ.dequeue();
14:    if (ExpE.distance < d_MAX) and (ExpE is not visited) then
15:    begin
16:      if((ExpE.U_turn_pos is empty and ExpE.Dflag == 1) or
17:              (ExpE.Dflag == 0)) then
18:        cur_KNN.add(ExpE.ObjList)
19:      else
20:      begin
21:        if (ExpE_r is not visited) then
22:          cur_KNN.add(ExpE.ObjList
23:                  ⋃ ExpE.U_turn_obj ⋃ Obtain_UExp_Obj(ExpE))
24:        else
25:          cur_KNN.add(Obtain_Exp_Obj(ExpE));
26:        for i := 1 to |ExpE_r.ExpEdge| do
27:        begin
28:          ExpE_r.ExpEdge[i].distance := ExpE_r.ExpEdge[i].distance
29:                  + 2*dist(ExpE.S, ExpE.U_turn_pos);
30:          Edge_ExpQ.enqueue(ExpE_r.ExpEdge[i]);
31:        end;
32:      end;
33:      for i := 1 to |ExpE.ExpEdge| do
34:      begin
35:        ExpE.ExpEdge[i].distance := ExpE.ExpEdge[i].distance
36:                + ExpE.Length;
37:        Edge_ExpQ.enqueue(ExpE.ExpEdge[i]);
38:      end;
39:      Final_KNN := cur_KNN;
40:      if |Final_KNN| >= K then
```

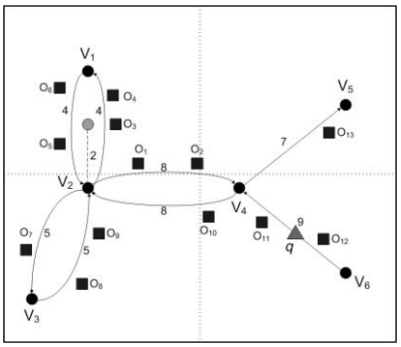
Fig. 5. EB_KNN_algorithm.

B. K Nearest Neighbor Query Processing

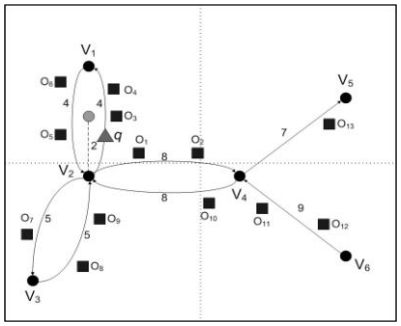
We propose the EB_KNN_algorithm (Edge-Based K Nearest Neighbor query algorithm) to process *K*NN queries in which data objects of interest are static and locate at the sides of the road. 0 shows our method for processing *K*NN queries using the EBNA index.

0-(a) shows an example that the query object locates on the Type 1 road. We consider the query issues a 4NN query. The data object O_{11} is obtained from function **Obtain_Dirobj** and is stored in array *KNN*. The links of the expansion edge information of (V_4, V_5) and (V_4, V_2) are inserted into *Edge_ExpQ* with the distance from the query object to the start point of the expansion edge for edge expansion. Therefore, *Edge_ExpQ* contains {(V_4, V_2), (V_4, V_5)} and the *KNN* contains {O_{11}}. Because the number of *KNN* is less than 4, we have to execute the edge expansion. We execute function *Edge_Expansion* to do edge expansion. We dequeue *Edge_ExpQ* to obtain the expansion edge information (V_4, V_2). Data object O_{10}, O_2, and O_1 are inserted into *cur_KNN*. The expansion edges of (V_2, V_1,) and (V_2, V_3,) are inserted into *Edge_ExpQ*}. Since V_2V_4 is the Type 3 road, we also have to consider edge expansion of the start point of edge (V_4, V_2). Because (V_4, V_2) has already visited and (V_4, V_5) has already existed in *Edge_ExpQ*, we do not insert this two edge into *Edge_ExpQ*. Now, *Final_KNN* is equal to *cur_KNN*. Since the number of data objects in *KNN* is equal to 4, the d_{MAX} is set to the distance from the query object to the data object O_1 (= 11). However, d_{MAX} is larger than the distance

from the query object to the start point of the edge (V_4, V_5), we continue to execute edge expansion. The data object O_{13} is inserted into *Final_KNN*. The array *Final_KNN* contains objects O_{11}, O_{10}, O_2, O_1, O_{13}. Because the distance from the query object to the start point of the expansion edge is larger than d_{MAX}, the edge expansion terminates, and we select the first 4 data objects O_{11}, O_{10}, O_2, O_1 as the query result. The *K*NN query processing terminates. 0-(b) shows an example that the query object locates on the *Type 3* road. We also consider the query issues a 4NN query. The data objects are obtained from *ObjList*. Since the query can reach anywhere of the road, we have to execute the edge expansion of the start point of the edge and the end point of the edge. Therefore, *Edge_ExpQ* contains {(V_2, V_1), (V_2, V_3), (V_4, V_2), (V_4, V_5)} after we visit the edge (V_2, V_4). We execute edge expansion according the order of *Edge_ExpQ*, then we obtain data object O_1, O_2, O_{10}, and O_3 as the 4NN result (see Fig. 6).



(a)



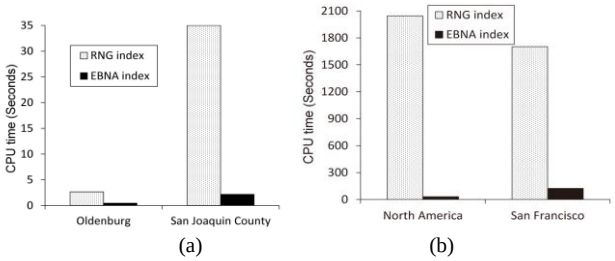
(b)

Fig. 6. One query on: (a) Type 1 road; (b) Type 3 road.

IV. Performance

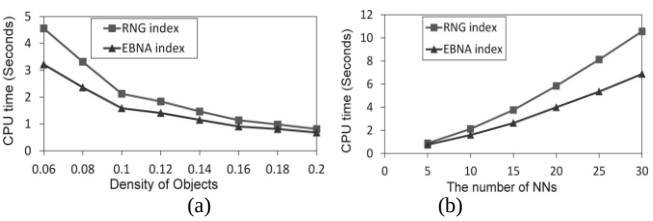
In this section, we study the performance of the *K*NN query processing using the RNG index structure and the EBNA index structure. We compare the two methods using the CPU time. The simulation is performed with the following variable parameters. Parameter *QTV* is the threshold value of the number of the quad-tree's leaf node. Parameter *NTV* is the maximal number of the NA-tree's leaf node on the EBNA index structure. Parameter *DOB* is the density of data objects in the space. Parameter *QNN* is the number of nearest neighbors for each query. Parameter *RD*1 is the ratio of the road of *One-Way Traffic Road*. Parameter *RD*2 is the ratio of the road of *Two-Way Traffic Road*. Parameter *RD*3 is the ratio of the road of *Two-Way Traffic Road with U-turn*. Parameter RD4 is the ratio of the road of *Arbitrary Traffic Road*. We will set different ratio of the different road types to compare the performance of the query processing on the RNG index structure and on the EBNA index structureeither.

We compare the performance of the construction of the RNG index structure and the EBNA index structure. 0-(a) shows the simulation result of the comparison of the construction of the RNG index and the EBNA index on the real-life dataset of *Oldenburg* and *San Joaquin County*. The performance of the EBNA index structure is better than that of the RNG index structure. When the number of vertices and edges increases, the execution time also increases. A comparison of the execution time of constructing the index structures of real-life datasets of *North America* and *San Francisco* is shown in 0-(b). We also find out that the performance of the EBNA index structure of these two real-life datasets is better than that of the RNG index structure. However, in these two real-life datasets, when the execution time of constructing the RNG index structure decreases, the execution time of the EBNA index structure increases. In 0-(a), the difference of the execution time of *K*NN queries of these two methods increases when the object density decreases. When the object density decreases, the number of edges which must be visited to obtain the interested data objects increases. In the RNG index structure, times of visiting the tree increases, if the number of edges have to be visited. In the EBNA index structure, the leaf node's links consist a graph. It only has to visit the NA-tree from the root to the leaf node one time. Then, according to the leaf node's links, it can obtain all the data objects that the query interests. Therefore, the search time decreases. Next, we experiments the performance of the different number of NNs in 0-(b). In order to obtain a large number of interested objects, the number of edges has to be visited increases. The condition makes times of visiting the quad-tree from the root to the leaf node increase in the RNG index structure; therefore, the execution time of *K*NN queries increases. However, in the EBNA index structure, it visits the NA-tree form the root to the leaf node only one time since it can obtain all the interested objects from the graph that leaf node's forms. The execution time of *K*NN queries of our method is shorter that Lu *et al*.'s method (see Fig. 7-Fig. 8).



(a) (b)

Fig. 7. A comparison of the execution time of real-life datasets of: (a) Oldenburg and San Joaquin County; (b) North America and San Francisco.



(a) (b)

Fig. 8. A comparison of the execution time of *K*NN queries of real-life datasets of: (a) San Joaquin County of different object densities of DOB; (b) San Joaquin County of different number of NNs.

V. Conclusions

We have proposed a road network model that captures the real-life road networks. Based on our proposed model, we have proposed an EBNA index that is an edge-based index structure. The EBNA index structure has made the *K*NN queries in real-life road network databases efficient. When a *K*NN query is executed, it only has to visit the root one time. We have concluded that the execution time of different real-life datasets of the EBNA index is faster than that of the RNG index. Second, we have shown the comparison of the execution time of *K*NN queries of the RNG index structure.

Acknowledgment

This research was supported in part by the Ministry of Science and Technology of Republic of China under Grant No. MOST 103-2221-E-110-056.

References

[1] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *Proc. the 21st International Conference on Data Engineering*, pp. 631-642, 2005.

[2] Y. G. Zou and Q. L. Fan, "OQ-Quad: An efficient query processing for continuous k-nearest neighbor based on quad tree," in *Proc. the 4th International Conference on Computer Science and Education*, pp. 197-202, 2009.

[3] W. Lu, Y. Shen, S. Chen, and B. C. Ooi, "Efficient processing of *k* nearest neighbor joins using MapReduce," in *Proc. the VLDB Endowment*, pp. 1016-1027, 2012.

[4] K. Xuan, G. Zhao, D. Taniar, and B. Srinivasan, "Continuous range search query processing in mobile navigation," in *Proc. the 14th IEEE International Conference on Parallel and Distributed Systems*, pp. 361-368, 2008.

[5] K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann, "Probabilistic range queries for uncertain trajectories on road networks," in *Proc. the 14th International Conference on Extending Database Technology*, pp. 283-294, 2011.

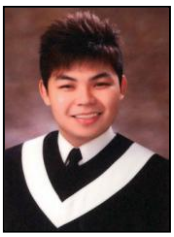
[6] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *Proc. the 29th International Conference on Very Large Data Bases*, pp. 802-813, 2003.

[7] Y. Lu, B. Cui, J. Zhao, H. Lu, and J. Shen, "Towards efficient and flexible KNN query processing in real-life road networks," in *Proc. the 9th International Conference on Web-Age Information Management*, pp. 230-237, 2008.

[8] Y. I. Chang, C. H. Liao, and H. L. Chen, "NA-Trees: A dynamic index for spatial data," *Information Science and Engineering*, vol. 19, no. 1, pp. 103-139, January 2003.

Ye-In Chang was born in Taipei, Taiwan, R.O.C. in 1964. She received her B.S. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 1986. She received her M.S. and Ph.D. degrees in computer and information science from Ohio State University, Columbus, Ohio, in 1987 and 1991, respectively. From August 1991 to July 1999, she joined the Faculty of the Department of Applied Mathematics at National Sun Yat-Sen University, Kaohsiung, Taiwan. From August 1997, she has been a professor in the Department of Applied Mathematics at National Sun Yat-Sen University, Kaohsiung, Taiwan. Since August 1999, she has been a professor in the Department of Computer Science and Engineering at National Sun Yat-Sen University, Kaohsiung, Taiwan. Her research interests include database systems, distributed systems, multimedia information systems, mobile information systems and data mining.

Meng-Hsuan Tsai received her B.S. degree in information management from Chang Jung Christian University in 2007 and his M.S. degree in Computer Science from National Pingtung University of Education in 2009. She is currently a Ph.D. student in Department of Computer Science and Engineering at National Sun Yat-Sen University. Her research interests include database systems and data mining.



Wu-Lun Wu received his B.S. degree in computer science and information engineering from Tamkang University in 2009, and his M.S. degree in computer science and engineering from National Sun Yat-Sen University in 2011. He is currently a system design engineer in Taiwan.