International Journal of Hybrid Information Technology
Vol.8, No. 5 (2015), pp. 177-186
http://dx.doi.org/10.14257/ijhit.2015.8.5.20

A Vor-KVGQ Index Structure of Mobile Objects Based on CoveRed Area

Zeng Ziwei[1], Ma Chi[*1, 2], Huang Yubin[3] and Wei Jing[1]

[1]*Department of Software, Liaoning Science and Technology University, Anshan 114051, China*
[2]*Dongling School of Economics and Management, University of Science and Technology Beijing, Beijing, China*
[3] *Anshan Iron and Steel Group Corporation, Anshan, China*
asmachi@126.com

Abstract

In recent years, the rapid development of high technologies, led the emergence of a variety of new applications. With the rapid development of mobile computing, wireless communication technology and deployment of a large number of location based service devices, making location based services have entered people's daily life. Nowadays, mobile telephone, laptop (PDA), Car GPS and other electronic products provide location based services to achieve the related functions. But the popular of the devices that support LBS function require LBS more popular and diverse, in order to better support the query for position information of mobile objects, need to propose more efficient index structures, the Vor-KVGQ index structure of mobile objects with keyword that combine grid Quad tree(Keyword Virtual Grid Quad tree, KVGQ) with Voronoi diagram based on covered area is one of them.

Key words: *mobile objects, LBS, covered area, space index structure*

1. Introduction

With the continuous development of high technology, high technology has entered people's daily lives, a lot of location-based service (Location Base Service, LBS)[1-5] equipment are deployed. The electronic products like mobile phones, laptops and car PC have achieved a function of location-based services. For example, in the road network, a car that traveling on the highway uses GPS system to query the nearest exit to "downtown Shenyang"; in the digital battlefield, a tank queries the nearest "enemy "headquarters; a pedestrian queries the nearest "11 Road "bus station or" Tesco "supermarket and so on. In order to better support the query to the mobile objects' position information, proposes an efficient index structure for moving objects is imminent.

2. Building the Data Model

Moving objects' management is managing moving objects' location information, the moving objects' position changes with time changes. When the position changes, the device send the current position information to the server; when a moving object sends a query to the server, the server receives the request and returns the results to the client.

Building the data model is depending on moving objects' specific properties, it will adopt different index structure depends on different properties. A construction for a moving object has the keyword information is same. In this paper, the current position index for the moving objects in two-dimensional space based is adopted. In order to achieve the construction of the moving object model, it needs to abstract the moving object-related information into elements, in this paper, each mobile object is defined a

ISSN: 1738-9968 IJHIT

unique identifier *Oid* to identify the position of moving objects, *Kid* to identify the moving object contains key information, *d* to identify coordinates information of the moving object, *v* to identify the speed information of the moving object, which is a vector, only has the size but not direction. In this paper, it adopts a four-tuple <*Oid*,*Kid*,*d*,*v*> to describe the position information of moving objects at a time. For other applications of the same type, the index method can also be applied in a multidimensional space.

3. Key Technologies

3.1. Grid Index

Grid file [6-7] is a typical and direct index structure like hash table. Grid file index is dividing the studied area to a number of fixed and same grids with horizontal and vertical lines; each grid cell corresponds to a data barrel, the barrel records spatial entities that each grid contains, such as point. Pointer to an object is placed in the appropriate barrel. A data barrel can often contain several adjacent cells. When queries an object with using grid index, finds the corresponding data barrel firstly, then finds the pointer of the object in the data barrel in order to locate the object.

3.2. Virtual Grid Quad tree Index

Virtual Grid Quad tree[8-10] (Virtual Grid Quad tree, VGQ) is a combination of grid index, compressed Quad tree structure and hash table, it is a index method of moving objects in regional coverage, compressed Quad tree is a development of Quad tree[11]. VGQ index first uses grid index to divide the space to a lot of equal and small area, regards the moving objects as points are distributed into these areas, then finds the position in this grid according to the coordinate of the point; then establishes compressed Quad tree corresponding to the grid index, and tree leaf nodes correspond to moving objects.

3.3. Voronoi Diagram Index

Voronoi diagram [12-14] is an important computational geometry, also known as Thiessen polygons or Dirichlet map; it is composed by continuous polygon that composed by a group of straight line connecting two adjacent points. It can divide space into a plurality of unit regions based on the position of elements in an object collection, namely Voronoi region. Each Voronoi region in Voronoi diagram corresponds to a moving object, and it does not overlap between different regions, and constitutes a space regions, when queries, first finds the region of the moving object in Voronoi diagram, then does the appropriate queries.

3.4. Vor-KVGQ Index

(1)Index Idea: The idea of Vor-KVGQ index structure is: First, uses the advantage of the grid file index that can quickly locate position, divides the space area into $N \times N$ grid cells, and then regards each cell as a entirety, if moving objects exist in the grid cell, adds the cell to the compressed Quad tree index; Conversely, if moving objects does not exist in the grid cell, you do not need to add the cell to the compressed Quad tree index. By this process, you can just index the area objects exist, greatly save storage space. Couples with the support of Voronoi diagram for static environments nearest neighbor's query , combines their advantages, in connection with the area that has little grids, the grid cells are abstracted into point objects and stored in the Voronoi diagram in order to better support nearest neighbor queries.

(2)Basic Structure: Vor-KVGQ index structure consists of three important hash table structures: Object table, Caddress table and Voronoi table. Object table stores the properties of moving objects, stores location information of moving objects , including unique identifier *Oid* of moving objects, the grid cell Cid which moving objects exist, the spatial coordinate *Oxy* of moving objects, the keyword information *Kid* that moving objects contain, we can use <*Oid*, *Cid*, *Oxy*, *Kid*> to express. *Oid*, *Oxy* and *Kid* can be got from location-based service applications; *Cid* is calculated according to Equation 1[15]:

$$Cid = (\mathrm{int})(hY * n - y) / hY * n + (\mathrm{int})x / hX \quad (1)$$

Where *x* and *y* are the spatial coordinates of the moving objects, *hX* and *hY* indicate horizontal width and vertical width of grid cell, n is the number of grid elements of the horizontal direction or the vertical direction, it should be noted that coordinate information in this paper is two-dimensional. Caddress table stores a corresponding relationship between grids and the node of compressed Quad tree. It has a unique identifier *Cid* of grid cells, the center position coordinates *Cxy* of the grid cells, the leaf node's address *Pleaf* of moving objects, can be expressed by <*Cid*, *Cxy*, *Pleaf*>. When inserts a grid into a compressed Quad tree, needs to insert the corresponding leaf node address of the grid into the hash table. Voronoi table stores the Voronoi diagram [16] generated by object abstracted into point within the region. It has the number *Vi*d of the object that abstracted into point within the region, it has the same value with the number of grid cells Cid, the corresponding coordinate *Vxy* that grid cells abstracted into a position, the information of Voronoi polygon Vor corresponding to the point, can be expressed by <*Vid*, *Vxy*, *Vor*>. Intermediate node (Nodepage) is a intermediate node of compressed Quad tree, it has the lower left corner coordinates min of the rectangular frame range, the top right corner coordinate max of the rectangular frame, the center coordinates center of intermediate nodes, the pointer pointing to its parent node, the pointer array children pointing to his four children nodes, the type childtype of their four children node (intermediate node or leaf node), the depth *level_id* that node pressed into the compressed Quad tree, the list *node_klist* that points the keyword information of intermediate node, can use <*min*, *max*, *center*, *parent*, *children*, *childtype*, *level_id*, *node_klist*> to express[4].

(3) Data Manipulation

(a). Insert: In Figure 1, when a new moving object or the object already exists moves t o a new grid cell, they all need to move the object into the grid.

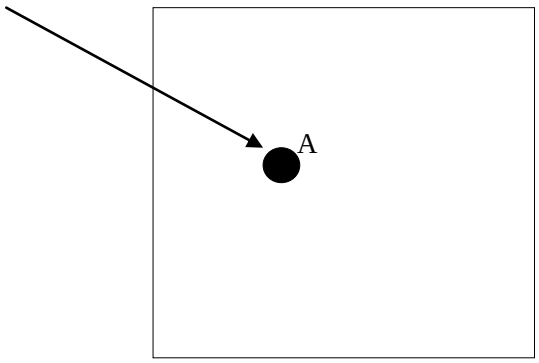


Figure 1. Insert

The algorithm is as follows:

Algorithm for Vor-KVGQ Insert

Input：Vor-KVGQ index structure, information of a moving object that are new inserted record
Output：a new Vor-KVGQ index structure
1.Calculates value *Cid* of the grid cell according to moving object's coordinate;
2. Finds the *Oid* of a moving object in table according to Cid;
3. if (*Oid* is not null)
4. Updates the value of *Cid* and *Oxy* where the row that *Oid* exists in Object table;
5. else
6. Inserts the relational information<*Oid*, *Cid*, *Oxy*, *Kid*> of moving objects into Object table
7. end if
8. Finds the value of Cid in Caddress table;
9. if(*Cid* is not null)
10. Gets corresponding node's address *pleaf* according to *Cid*;
11. Examines whether the list *leaf_kli*st the pleaf node points includes *Kid*;
12. if(*Kid* is null)
13. Inserts *Kid* into *leaf_klist* and put the relational value of *oid_num* as 1.
14. else
15. Adds 1 to *Kid* relates to *oid_num*， insert *oid_list* into *leaf_klist*;
16. end if
17. else
18. Initializes the center coordinate *Cxy* and leaf of *Cid*;
19. Abstracts grid cell into point and inserts into compressed Quad tree. then return leaf node's address;
20. Inserts new information into Caddress table;
21. Updates Voronoi diagram in this region;
22. end if
23. End

(b). Delete: In Figure 2, when a moving object moves to another grid from the current grid or disappears from the system, it needs to remove the object from the grid cell, also needs to update the object-related data in the index structure, including the information of Object tables, Caddress tables, Voronoi tables and the leaf nodes of compressed Quad tree.

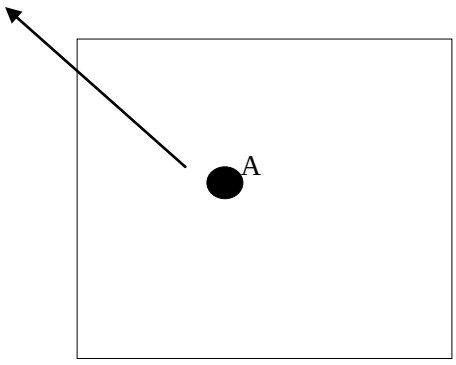


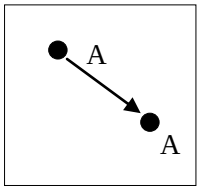
Figure 2. Delete

The algorithm is as follows:

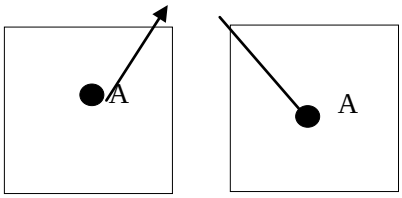
Algorithm for Vor-KVGQ Delete

Input：Vor-KVGQ index structure，delete a moving object record
Output：the Vor-KVGQ index structure after deletes record
1. if(moving objects are not null)；
2. Finds the relational *Oid* in Object table and delete data's information;
3. Finds the relational leaf node pleaf in Caddress table according to *Cid*；
4.Find *Vid* from leaf nodes relares to *leaf_klist*;
5. Deletes *Oid* from *oid_list*;
6. Minuses 1 to *oid_num*；
7. if(*oid_num*==0)
8. Deletes the data that keyword is Vid in *leaf_klist*；
9. Deletes the data relates to *Cid* in Caddress table；
10. Deletes the relational leaf nodes in compressed Quad tree；
11. Deletes the relational generative points in Voronoi diagram;
12. end if
13. else
14. Target object does not exist；
15. end if
16. End

(c). Update: In Figure 3, dues to the moving object moves continuously, the position o f a moving object will change with time changes. The updated operations for moving obje cts are divided into two situations: First, if the position of the moving object changes in a grid cell, only needs to update the location information of moving objects; second, if the moving object moves from the current grid cell to an another grid cell, it will cause chang es in the index structure.



(a) The Position of the Moving Object Changes in a Grid Cell



(b) The Moving Object Moves from the Current Grid Cell to an Another Grid Cell

Figure 3. Update

The algorithm is as follows:

Algorithm for Vor-KVGQ Update

Input：Vor-KVGQ index structure，update the position of a moving object record
Output：the Vor-KVGQ index structure after updates
1. Calculates NewCid according to the coordinate of the moving object；
2. Finds *Cid* from Object table according to *Oid*；
3. if(*NewCid*!=*Cid*)；
4. Callings the algorithm 2.2, delete *OldCid* relates to *Oid* from Object table, Caddress table, Voronoi table and index structure;
5. Callings algorithm 2.1 and insert *NewCid* relates to *Oid* into Object table,Caddress table Voronoi table and index structure;
6. else
7. Updates data *Oxy* relates to *Oid* in Object table；
8. end if
9. End

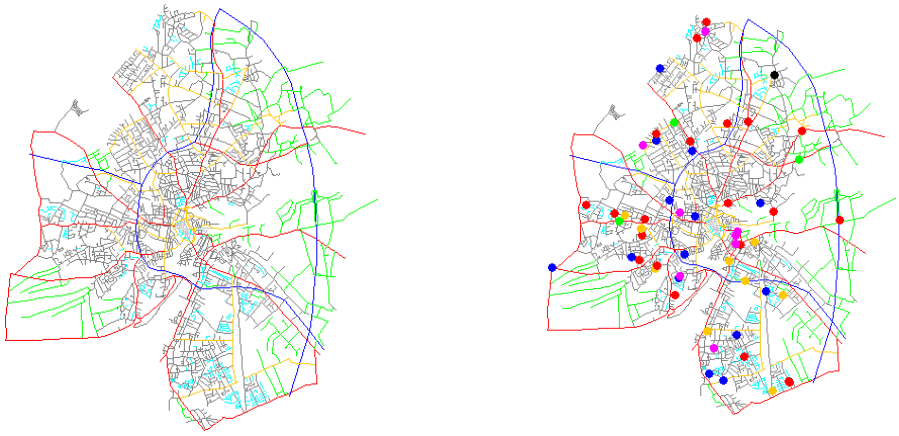
(4). Find: Many location-based services applications requires to implement the moving object's position information query, the operation is to locate the position of moving objects in order to determine the location-related information of moving objects. The find operation for moving objects can be divided into two situations: one is to query the location information of specified moving objects; another is to query moving objects within specified area. For the first case, it can use Caddress table to find the leaf nodes relates to moving objects in compressed Quad tree directly, and then return to the location-related information; For the second case, it needs to do recursive traversal for compressed Quad tree through scissors rules, find the space area intersects with the given region, then find the spatial region of the moving objects and compare to specified moving objects whether the moving object is found.The algorithm is as follows:

Algorithm for Vor-KVGQ Find

Input：query the initial node *pNode*, query node *q*.
Output：the leaf nodes of compressed Quad tree
1. if(*q* is a query type of point)
2. Gets *Cid* from Object table;
3. if(*Cid* is not null)
4. Returns the leaf node leaf in compressed Quad tree relates to *Cid*；
5. else
6. returns the target object is not exist；
7. end if
8. else
9. for (every leaf node of *pNode*)
10. if(it is a leaf node and intersects with queried region)
11.Filters the moving objects of grid cells and finds the target object；
12. else
13. (it is a middle node and intersect with queried region)
14. search(*children*, *q*)
15.end if
16. end for
17. end if
18. End

4. Experimental Design and Analysis

In the experiment, in order to reflect movement and distribution of moving objects in the real road network, the experimental data set this paper uses is taken from the literature [17] that is moving object data generator based on the road network, currently most of moving object theories adopt the data generator as their experimental data set, the generator uses German Oldenburg city's map as input, generates a series of data related to the city road network by generator as output. The road network includes various sections and intersections, each time requires reporting all position information and velocity information of moving objects, the generator generates a random distribution of moving objects and they have their own movement routes.



(a) Before Data Generation **(b) After Data Generation**

Figure 4. The Map of the Road Network

In Figure 4 (a), it is a map of the road network before the data is generated, and figure 4 (b) is a map after the road network data is generated, the dot in this map represents different colored moving object, in order to make the query points in the experiment are in accordance with the moving objects' randomness in real life, this experiment randomly selects some moving objects as the nearest neighbor query requester, makes nearest neighbor query points distributed into network uniformly.

4.1. Experimental Program

In order to reflect the real performance and nearest neighbor query's efficiency of the index structure, we do a comparative based on the mentioned experimental platform in this paper to test its own performance of the index structure and two nearest neighbor queries' performance based on this index structure. Specific parameter settings are shown in Table 1:

Table 1. Experimental Parameters

parameter	range	default
geographic region	20000	none
the number of	10000	10000
the number of	0~500	100
threshold	[0.1,	0.2
division of the grid	0~300	150
the time interval	5	none
dataset	Netwo	none

4.2. Experimental Data and Analysis of Results

Figure 5 is the response time when establish a Vor-KVGQ index structure that the data of moving objects are 20000 and 60000 under different grids. As can be seen from the figure, the Vor-KVGQ index structure's change is not obvious with the number of underlying grids are gradual increase, when partition is greater than 200, the building time of the index structure is basically unchanged, the main reason is the affect to index structure's building is small when the number of grid reaches a certain number. We can see that different number of moving objects has also affect to the time of index structure's building, when the number of moving objects increases by 3 times, the building time of index structure also increases by 3 times. The results can be drawn as Vor-KVGQ index structure has a good scalability and flexibility for dealing moving objects.

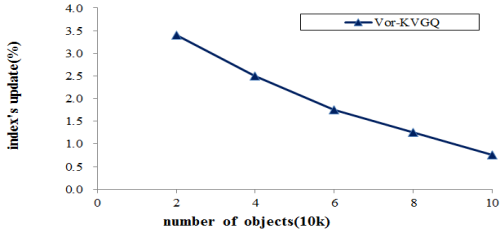


Figure 5. Different Grids and Number of Moving Objects' Effect to the Building Time of Index Structure

Figure 6 is the updated number's compared between index structure and moving objects in 5s when the number of moving objects changes from 10000 to 100000.We can know from the Figure 4.3 that with the increase of number of objects, not only the number of update is not increase, but also shows a downward trend. When the number of moving objects changes form 20000 to 40000, the number of update reduces fastly, when the number of moving objects changes from 60000 to 100000, although reduced speed becomes slowly, but it also reduce the number of update. According to the experiment's result, Vor-KVGQ index structure is an effective structure.

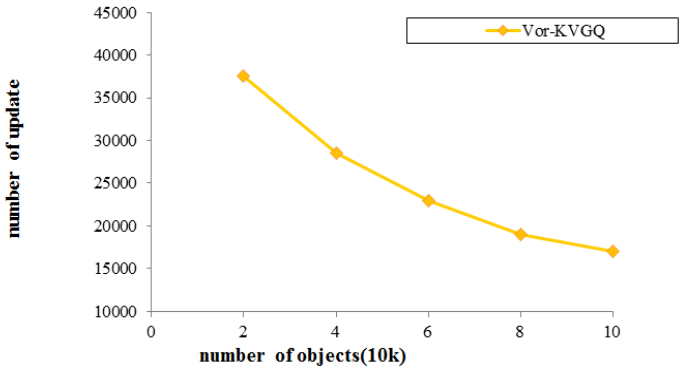


Figure 6. The Updated Number's Compared of Different Dataset

Figure 7 is the effect to the building time of index structure for different dataset. As see form Figure 7, we can know with the increase of grid's number, the building time of index structure is not increased quickly. When the grid's number becomes form 50 to 150, the change of curve is obvious, when the grid's number is larger than 150, it has not change. We can also see that when the grid's number is 50 and moving objects' number is 20000, Vor-KVGQ index structure's building time is 0.04s, but Vor-KVGQ index structure's building time is 0.14s when the moving objects' number becomes 60000.

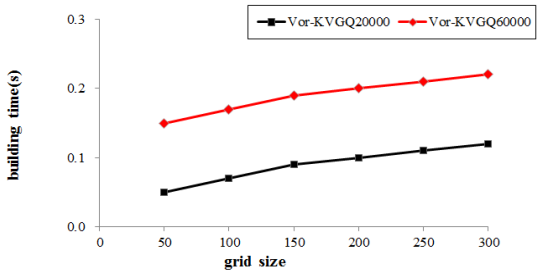


Figure 7. The Number of Update's Effect to Different Dataset

According to Figure 6 and 7 we can know, the update of the index structure has closely relation to the distribution of moving objects in traffic's network. When the traffic's network has a large number of moving objects, the whole space area will be covered by grid cells after the index structure is established completely. If the moving objects' position information changes, we can only update the relevant grid cell that has moving objects, does not need to update the index structure. Therefore, Vor-KVGQ index structure is a relatively stable index structure.

Figure 8 is the response time of Voronoi diagram between different thresholds and different dataset. As can be seen from the figure, the building time of Voronoi diagram between 50000 moving objects and 100000 moving objects is almost the same. The number of moving objects has a little effect to the Voronoi diagram's building time, but the threshold has a big effect on the building time of Voronoi diagram. With the threshold value increases, Voronoi diagram's building time gradually increases. When the threshold becomes from 0 to 0.8, the building time of Voronoi diagram changes fast, when the threshold is larger than 0.8, it is not change. According to the result, 0.8 is a best value.

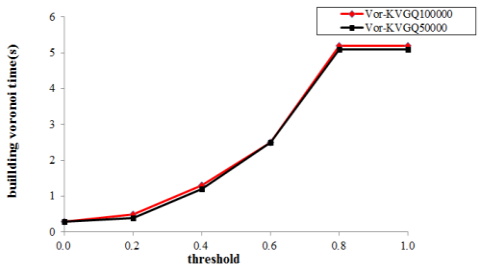


Figure 8. The Effect of Voronoi Diagram's Building Time to Different Threshold

5. Summary

Currently, location-based services are becoming more popular, people's demand on the location query of these applications increases complex. In order to better do location query operation, this paper proposes a Vor-KVGQ index structure based on region covered, gives the thought of the index structure, specific algorithm of insert, delete, update and query operations, and designs an experiment from three aspects of the meshing, dataset's size and threshold's size, carries out a detailed analysis of the experimental results and obtains Vor-KVGQ index structure is efficient.

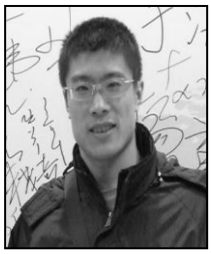
References

[1] T. Roza and G. Bichev, "An overview of location-based services", BT Technology Journal, vol. 21, no. 1 **(2003)**, pp.20-27.
[2] E. Kaasinen, "User needs for location-aware mobile services", Personal and Ubiquitous Computing, vol. 7, no.1, **(2003)**, pp.70-79.

[3] D. Zhonghua, P. Yong and Z. Tianyu, "LBS system security", Journal of Tsinghua University (Science an d Technology), vol. 10, no.1, **(2011)**, pp.3-6.
[4] T. Cho and S. Choi, "A Multi-path Hybrid Routing Algorithm in Network Routing", International Journal of Hybrid Information Technology, vol. 5, no. 3, **(2012)**, pp.41-46.
[5] L. Guohui and Z. Xiya, "Indexing Moving Objects Trajectories on Fixed Networks", Journal of Computer Research and Development, vol. 43, no. 5, **(2006)**, pp. 828-833.
[6] F. Huiyan and G. Junfeng, "A continuous nearest neighbor queries in road network", Computer Engineeri ng,vol. 8, no. 1, **(2010)**, pp.5-8.
[7] Z. Jingmin, W. Peichong and Lu Fengjia, "Indexing method of moving objects on networks", Computer Engineering and Applications, vol. 45, no.12, **(2009)**, pp.144-146.
[8] E. Kaasinen, "User needs for location-aware mobile services", Personal and UbiquitousComputing, vol. no.1, **(2003)**, pp.70-79.
[9] C. Yuhong and S. Lei, "Explore of the spatial data index technology based on R-tree", Computer Applica tions and Software, vol. 25, no.12, **(2008)**, pp.169-179.
[10] D. Hongyan, W. Feng and Z. Zhao Qian, "A R-tree index structure for multi-scale representation of spati al data". Chinese Journal of Computers, vol. 32, no.1, **(2009)**, pp.177-184.
[11] D. Eppstein, M. T. Goodrich and J. Z. Sun, "The skip Quad tree: a simple dynamic data structure for mul ti-dimensional data", In Proc. Of the 21th Annual Symposium on Computational Geometry, **(2005)**, pp. 2 96-305.
[12] L. Song and H. Zhongxiao, "The study of reverse nearest neighbor based on Voronoi diagram", Journal o f Harbin Engineering University, vol. 3, no.1, **(2008)**, pp.8-11.
[13] Z. Wang and Q. Niu, "Research on Hybrid Query Expansion Algorithm", International Journal of Hybrid Information Technology, vol. 5, no. 2, **(2012)**, pp.207-212.
[14] Y. Ze-xue and H. Zhong-xiao. "Continuous Reverse nearest Neighbor Search Based on Voronoi Diagram ". Computer Engineering, vol.40, no. 1, **(2014)**, pp.272-274.
[15] L. Jiajia, "Research and Implementation of Range Departure Monitoring Algorithms for Groups of Movi ng Objects Based on Covered Area", **(2010)**, Shenyang: Northeastern University.
[16] Q. Jingwei, "The study and implement of mobile objects' K nearest neighbor query based on covered are a", Northeastern University, **(2011)**, pp. 37-67.
[17] B. Yao, R. F. Li and P. Kumar, "Reverse farthest neighbors in spatial databases", In Proc. of the 25th IEE E International Conference on Data Engineering,**(2009)**, pp. 664-675

Authors

Zeng Ziwei has received his master's degree At the Northeastern University in 2001. Since 2010 he has been working as Professor in School of Software Engineering at University of Science and Technology Liaoning. His research interests include Moving Objects Database, wireless sensor networks and Ad hoc network.



Ma Chi has received his Ph.D. in Dangling School of Economics and Management at the University of Science and Technology Beijing. Since 2010 he has been working as Associate Professor in College of Software at University of Science and Technology Liaoning. His research interests include pattern recognition and data mining.

Huang Yubin is currently pursuing his master's degree in College of Software at University of Science and Technology Liaoning. Since 1998 he has been working as Senior Engineer in Anshan Iron and Steel Group Corporation. His research interests include pattern recognition and data mining.

