OQ-Quad: An Efficient Query Processing for Continuous K-Nearest Neighbor Based on Quad Tree

Yong-Gui Zou
Sino-Korea Chongqing GIS Research Center,
College of Computer Science,
Chongqing University of Posts &Telecom.,
Chongqing, China
zouyg@cqupt.edu.cn

Qing-Lin Fan
Sino-Korea Chongqing GIS Research Center,
College of Computer Science,
Chongqing University of Posts &Telecom.,
Chongqing, China
qinglinfan@gmail.com

***Abstract*—With the growing popularity of mobile computing and wireless communications, managing the continuously changing information of the moving objects is becoming feasible, especially in LBS application which is characterized by a large number of moving objects and a large number of continuous queries. In this paper, we focus on continuous k-nearest neighbor (CkNN for short) query and propose a query method based on a quad tree to support continuous k-nearest neighbor query for moving objects, in which the main idea is to use a quad tree to divide the static spatial space for the moving objects. In the interested region, we use the quad tree and hash tables as an index to store the moving objects. Then we calculate the distances between the query point and the moving objects from inside to outside to get the result. Our comprehensive experimentation shows that the performance of the proposed method is better in memory consumption and processing time than the CMP algorithm.**

***Index Terms*—Continuous Query; CkNN; Quad Tree; Index; Moving Objects; Spatial-temporal**

I. INTRODUCTION

In traditional spatial-temporal database, there are two fundamental categories of query: (1) k-nearest neighbor query which is to find the k neighbors that are nearest to the specified query point and (2) range query which is to find the objects within the query range. Recently, the continued advances of mobile computing technologies including GPS, the wireless sensors network and Internet-worked mobile devices have enabled a variety of new applications to become reality, such as traffic management, location-based services and so on. All these applications should monitor continuously the positions of moving objects [1, 2]. And the query may also be moving, which depends on the requirements of the users. So in these applications, the query can be divided into four categories: (1) stationary query on moving objects, (2) moving query on stationary objects, (3) stationary query on stationary objects and (4) moving query on moving objects. In this paper, we are mainly concerned with the last category of query in k-NN style.

The processing of CkNN query can be divided into two steps: initial processing and continuous update. In initial processing step, it takes a snapshot k-NN query to get the initial result. Then in the continuous update steps, the result of the query is updated whenever the locations of moving objects have been updated. There are two situations in which the result of the CkNN will be changed with the objects' movement: (1) some objects move near to the query point, which may make the query circle shrinking and (2) some objects move further away from the query point, which may make the query circle extending.

In previous research, a grid index is used to index the both moving objects and query continuous k-nearest neighbors so that the update message of the moving objects' location can be quickly detected by the query. However, almost all the previous grid index assumes that the movement of the objects is unconstrained and is independent of each other. Actually, in the real environment, the objects are all moving within spatially constrained in the road network, e.g. vehicles always move on road network, and trains on the railway network. Furthermore, the road situation may have effects on their speed and orientation; this also can help us perform the query operation efficiently.

In this paper we propose a new index structure which is based on a quad tree for CkNN query processing. In this way, the processing time consumption is decreased greatly than the traditional grid indexing method. And the query indexing over this structure is based on the main memory, so the search time of the query processing is hence significantly decreased.

The rest of the paper is organized as follows. Part II surveys the related work. Part III present the system model and index structure based on a quad tree. Then Part IV describes the query process of CkNN using the grid structure presented in Part II. Performance valuation will be shown in Part V. Finally, conclusions are drawn in Part VI.

II. RELATED WORK

In the last decade, k-NN queries have fueled the spatial and spatio-temporal database community with a series of interesting noteworthy research issues. After the first algorithm for k nearest neighbor search over a moving query point was proposed in [3], another continuous 1-nearest neighbor query process method in road space is proposed in [4]. And k-NN queries are well studied in traditional databases [5, 6]. The main idea is to traverse a static R-tree-like structure which is

developed from R-Tree. However, all the k-NN processing methods based on the above algorithms only support the static data processing, which can not be extended to support the highly dynamic data. Later, in spatio-temporal databases, a direct extension of traditional techniques is to use branch and bound techniques for TPR-tree-like structures [7]. The TPR-tree family [8, 9] are used to index moving objects, giving their future trajectory movements. Although this idea works well for snapshot spatio-temporal queries, it cannot deal with continuous queries. Continuous queries need continuous maintenance and update of the query answer.

Continuous k-nearest neighbor queries (CkNN) are first addressed in [10] from the modeling and query language perspectives. As the application grows, kinds of query process methods for CkNN appear. The method using grid structure to divide the static space are proposed, such as SEA-CNN algorithm proposed by Xiong [11] and CMP algorithm proposed by Mouratidis [12]. CMP presents an incremental approach to support continuous query processing, which is to monitor the changes of the query results. The idea of this approach is to register the queries in its influenced region and try to get the new results based on all the moving objects in the influenced region. In this paper, we will propose a new method (named OQ-Quad), which is based on a quad tree, to support continuous k-NN processing. In [12], it has been proved that CMP outperforms YPK-CNN and SEA-CNN [11], so we only compare OQ-Quad with CMP. In the following part, we will detail the proposed OQ-Quad.

III. DATA STRUCTURE AND BASIC OPERATION

In this paper, we focus on processing continuous k-NN query with the quad tree index in main memory and assume that objects move in 2-D space. In this part, we will overview our index data architecture (named OQ-Quad Tree index) and present the method how to manage the moving objects.

A. OQ-Quad Index

The OQ-Quad index structure is shown in Fig.1 which includes three parts: Query table, Object table, OQ-Quad tree.

(1) Query table

The query table is a hash table for storing the query in the structure. A query Q is described as Q(QID, loc, kNN-dist, kNN-list), where QID is the identifier of the query, loc is the location of the query point, kNN-dist is the maximum distance between q and its k-th nearest neighbor, and kNN-list means the list of the k nearest neighbors.

(2) Object table

The object table is also a hash table for storing moving object. Its main attributes are OID and its location. This will provide the most efficient access to the coordinates of the moving objects.

(3) OQ-Quad tree

The invoker of CkNN query is also an object which is called query point, so the query can be abstracted as a moving object. We use the OQ-Quad tree to present the moving objects and moving query. The quad tree is constructed according to the previous grid. In the OQ-Quad tree, the leaf nodes are not in the same layer. Each node has two list, objects list and query list. Objects list which stores the identifiers of objects are used to update the quad tree and decide the identifier. The query list is used to support incremental query processing. In order to efficiently support intensive updates, these two lists are also implemented in the hash table.

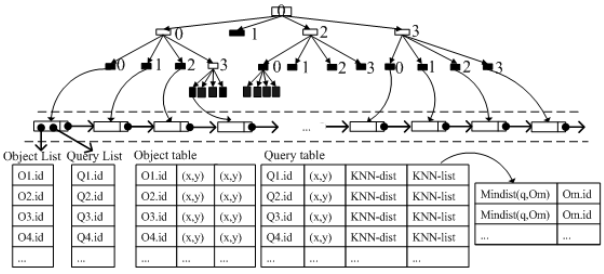


Figure 1. OQ-Quad Index

B. Operation

Update is an integrated operation in the OQ-Quad tree. For each moving object, its location is changing from time to time. That means each moving object will come out from one cell later into another cell frequently. While updating the coordinate of a moving object p in the object table, if p moves across the cells, p's identifier are deleted from the object list of the "old" cell and inserted in the object list of a ""new" cell. During each result update cycle, the object table, query table and the OQ-Quad tree will be updated firstly. When the object moves from one cell to another, the coordinate of the object table will be updated. After a new continuous k-NN query q registered, the k-NN search algorithm will be invoked to get the result of q, the value of the query table will also be updated. All the updates are performed partly in the follow cases.

(1) The query point is moving

When the query point is moving, as shown in Fig.2(1) and Fig2.(2), the kNN-dist may be changed and the query range will be extended., the distances from the query object to all the objects of all the related cells will be recomputed, and if the distance from the new object to the query point is less than the kNN-dist, the query result will be changed. It means that object O_3 will be deleted and O_7 will be inserted into the query result and the distance between q and O_7 becomes kNN-dist.

(2) The object is moving

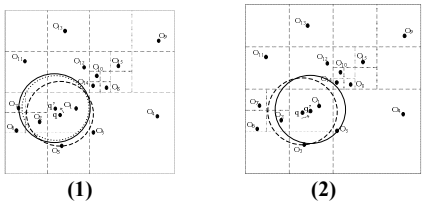
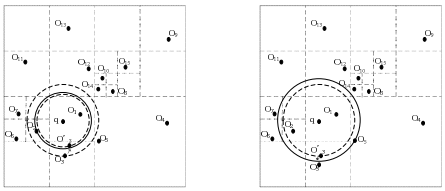


Figure 2. Example of operations when the query point is moving

As shown in Fig.3(1) and Fig.3(2), the kNN-dist is the distance between q and O_3. When O_3 is moving toward the query point, the kNN-dist and the cycle radius will be smaller. When O2 is nearer to query point q than O_3, O_2 will be inserted

978-1-4244-3521-0/09/$25.00 ©2009 IEEE

into the kNN-list and O_3 will be deleted. The query result will be changed. On the other case, O_3 is moving reversely to the query object q, and when the distance O_3 and q is larger than the distance O_5 to q, the result will also be changed.



(1) (2)

Figure 3. Example of operations when the object is moving

IV. CONTINUOUS K-NN QUERY PROCESSING IN MAIN MEMORY QUAD TREE

In this paper, we focus on processing continuous k-NN query in main memory quad tree index. It is assumed that all the objects move in 2-D space. Section A designs the update strategy and algorithms. Section B proposes an incremental query algorithm using the quad tree. Section C describes incremental query processing using quad tree.

A. Update Strategy and Algorithms

The updating of our index includes Quad tree updating and Hash table updating. For the Quad tree, the subtree is deeper for the high-density region. It is necessary to timely merge the corresponding nodes of the quad tree to improve the efficiency of the queries. For the hash table, when the new location of the object exceeds the old node's range, then update the hash table. We assume the structure information of a moving object is (OID, Loc_p, Loc_n), which the OID is the dentifier of the moving object, the Loc_p is the past location of the moving object and the Loc_n is the current location of the moving object. The process of updating the moving objects is as follows:

Algorithm1: Update the OQ-Quad tree Using (OID, Loc_p, Loc_n)

Input: OQ-Quad tree OQQT, Object Table OT

1) For each object o in Object table OT do

2) Using OT.o.Loc_p to find the old leaf node L_o

3) If(OT.o.Loc_n$\in L_o$.range) then

4) o.Loc_p=o.loc_n

5) Else then

6) delete o from L_o.Object list

7) If(L_o's father node.four leaf node.Object list.size( )<N) then

/* N is the max number of objects in one cell*/

8) Merging the four sub-leaf node of L_o's father node

9) Else then

10) Using OT.o.loc_n to find the new leaf node L_n.

11) Insert o into L_n.Object list

12) If (L_n.Object list.size( )$\geqslant$N) then

13) Splitting the leaf node L_n into four sub-leaf nodes

14) End for

B. Ck-NN Search Algorithm

The k-NN search algorithm is used to compute the results of the new registered and moving queries. The basic idea of the algorithm is to construct cell levels around the query point then use objects in the levels to refine the results of the queries until all the cells have been checked by the minimum distances from the query to the level. Above all, the algorithm tries to get the result by checking as few as objects as possible. As shown in Fig.4, the region is divided by the quad tree according to the number of objects in the cells. The cell containing the query point q is level 0. All cells around the level 0 construct level 1, all the cells around level 1 is level 2 and so on.

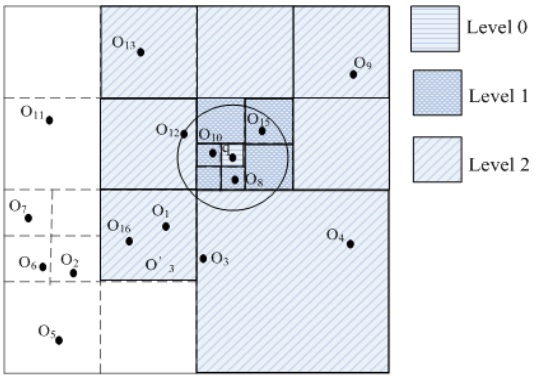


Figure 4. Example of level search

Firstly, if the total number of objects in the current levels is not larger than k, the algorithm will increase level search until the number of objects is not small than k. Then insert all the objects into the query list and calculate the distances between the query point and the moving objects to get the query result. For a given query, we make mindist(q, level i) denote the minimum distance between q to the level i, q.k-NN-dist denotes the minimum distance between query point q and its kth nearest neighbor. NO(Li) denotes the total number of objects in level i. In this paper we only sort the qualified cells, which contain at least one object. And the distance mindist(q, level i) must be larger than q.k-NN-dist, or the query result will be discarded before the increase search level. The query result won't be decided until the minimum distance from query point q to the outermost level is not less than q.kNN-dist.

Fig.5 explains the idea of level search based on the k-NN search algorithm. The circle area with the radius of q.kNN-dist is the influence region of query point, which is in the circle center. The 4-NN query q search procedure starts at Level 0. All cells of level 0 are the shadow part in Fig.4. Firstly, we obtain the initial candidates O_8,O_9,O_{10},O_{14} and set the distance between q and O_9 to be the q.k-NN.dist, dist(q,O_9), and the influence region is enclosed by dashed line. It is shown that dist(q,O_9) is larger than dist(q, level 0), so the algorithm increases the search level, visits the level 1 to get the exact

query result. All the objects in the level 1 will be computed. A better 4-NN result will be found and the influence region will be updated, which is encoded in real line. At the initial phase, our k-NN search algorithm avoids sorting the cells which must be accessed and the method of filtering cells reduces the cost of sorting and processing time. It comes true that our algorithm is better than the algorithm presented for CMP in [13].

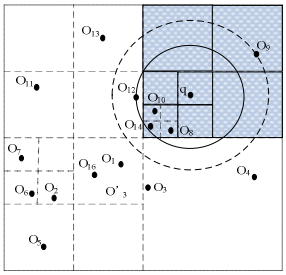


Figure 5. Example of 4-NN by level search

Algorithm2: k-NN Search (OQQT, QT, q)

Input: OQ-Quad tree OQQT, Query Table QT, k-NN Query q;

Output: q.kNN-list

1) q.kNN_list = Ø; /*Initialize k-NN*/

2) i = 0; /*Initialize cell level*/

3) q.kNN_dist = ∞; /*Initialize kNN-dist */

4) While (q.kNN_list.size() + NO(L_i) ⩽ q.k) do

/*NO(Li) is the No. of objects in search level i*/

5) Insert all objects in Li into q.kNN-list;

6) If (q.kNN_list.size() = = q.k) then

7) Update q.kNN-dist;

8) i = i + 1; /*Increase cell level*/

9) End While

10) MinLev_dist = mindist(q, L_i);

11). While ((q.kNN_list.size() < q.k) or

(MinLev_dist < q.kNN_dist)) do

12). For all cells c_j in L_i do

13). If ((c_j not empty) and ((q.kNN_list.size() < k)or

(mindist(q, c_j) < q.kNN_dist))) then

14). Get the minimum distance between c_j and L_i, mindist(q, cj);

15) Store all the values <c_j,(q, c_j)> into heap DH;

16) While DH is not empty do

17) If (q.kNN_dist < mindist(q, c)) then

18) Get the value <c, (q, c)> from DH;

19) For each object o in cell c do

20) If (q.kNN_list.size() < k) then

21) Insert o into q.kNN_list;

22) Else then

23) Update q.kNN_list and q.kNN_dist;

24) End For

25) Else then

26) Clear the values in DH;

27) End While

28) i = i + 1 /*Increase cell level*/

29) MinLev_dist = mindist(q, L_i);

30) End While

C. *Incremental Query Processing Using Quad tree*

In this section, we will overview the method of incremental query processing using OQ-Quad. We focus on two main aspects: the query point is static and moved. If the query is static, the idea of incremental query processing is to make full use of the last result to reduce the cost of query processing.

In OQ-Quad index, the query list is required to support incremental processing. All the cells overlapped by the influence region of the query point q, may influence the answer of q, when the objects in these cells moved. The identifier of q is inserted into the query lists of the influenced cells, and if any object in any one of the cells moves, the q's k-NN result will be updated. For example, if one object o moves from one cell Cold to another cell C_{new}, the identifier of o will be deleted from the object list of Cold, and the identifier of o will be deleted from all the q.kNN-list which belongs to C_{old}. The identifier of the object will be inserted into the object list of the new cell C_{new}, For each query q in the query list of C_{new}, if dis(o, q) is not larger than q.kNN-dist, the result of query may be updated. The results of all the moved objects in the queries' influenced region will be cleared, and will be reevaluated by the k-NN search algorithm.

We use only one heap to help processing all the queries, and keep the objects in the query in k-NN list. However, for CMP, every query has one stored heap and a visit list, and all the objects in these visited cells need to be re-checked. So the costs of memory and processing time in our algorithm are both better than that in CMP.

V. PERFORMANCE EVALUATION

In this part, we evaluate the performance of our method based continuous k-NN process algorithm (CkNN) and compare the performance of our algorithm with CMP.

The data sets in our experiment are generated by the same spatio-temporal data generator of [18]. Queries are randomly created locations and the moving object datasets are used as the location update message streams. The settings of system parameters are summarized in Table I. Here slow, medium, and fast objects speed are the default speed values in the generator [18]. In each experiment, only a single parameter is varied, and the others remain their default values. We finish all the simulations in a PC which contain a Pentium 3.2 GHz CPU with 1GByte memory.

 978-1-4244-3521-0/09/$25.00 ©2009 IEEE

TABLE I. THE SETTINGS OF PARAMETERS IN EXPERIMENTAL EVALUATION

Parameters	Default	Range
Number of moving objects	100k	10,30,50,100(k)
Number of queries	5k	1,3,5,10(k)
k	32	1,16,32,64
Speed of objects/query	medium	Slow, medium, fast
Location update rate of objects	50%	10,30,50(%)
Location update rate of queries	30%	10,30,50(%)
Moving object density in a cell	35	20,25,30,35,40,45

A. *Effect of index granularity*

Here the maximum number of moving objects in a cell and the population of the objects in the whole region are the key issues of this index structure. Because these two parameters will decide the height of the index tree, which will affect the processing time and memory consumption. As described before, all the moving objects are stored in the quad tree, so if the maximum number of moving objects in a cell is invariable and the population of the objects is become larger, the quad tree must be very higher, then processing time may be increased naturally; The tables which store the moving objects may be very large, the cost for update (including insertion and deletion) will be very large, as shown in Fig. 6.

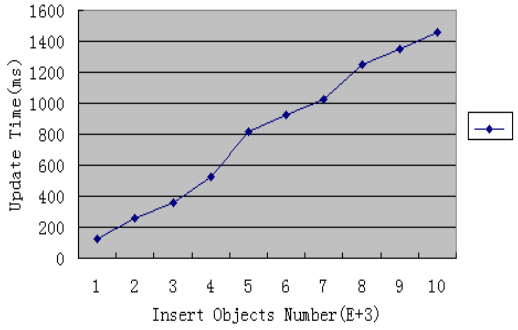


Figure 6. Overall update time versus insert objects number

B. *Effect of number of objects*

As the population of the objects gets larger, the time the algorithms take to process the messages gets longer. When the population of moving objects becomes larger, the number of objects in each cell will increase, and the size of cells will get smaller which will affect the queries, as shown in Fig. 7.

C. *Effect of location update rate*

The cost of query processing by our algorithm and CMP is naturally increased as the location update rate increased. It is shown in Fig. 8 that the performance of continuous k-NN query processing over moving objects decreases with the agility of objects increases. However, we can see that the performance of our algorithm still outperforms CMP under all settings.

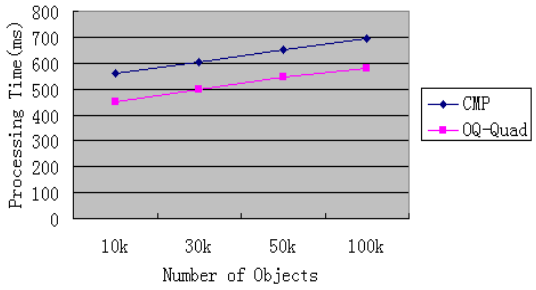


Figure 7. Overall processing time versus number of objects

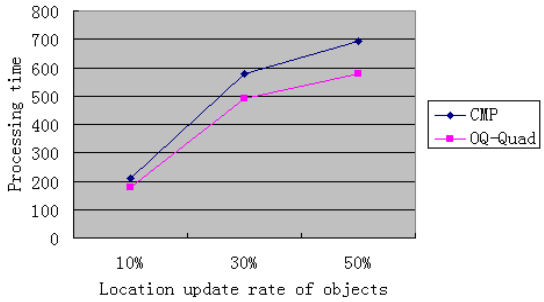


Figure 8. Overall processing time versus location update rate of objects

D. *Effect of k*

K gets bigger, the neighbors we need to find gets more, and it is natural that the processing time increases linearly. As shown in Fig. 9, with all sets, OQ-Quad outperforms CMP.

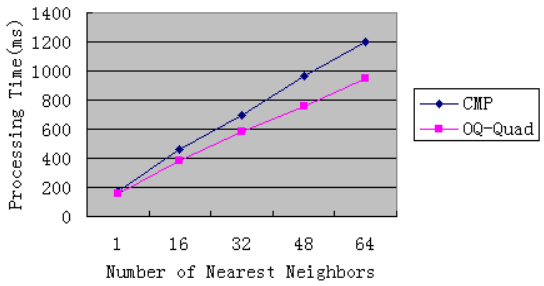


Figure 9. Overall processing time versus number of queries

E. *Effect of number of queries*

It is also natural that the processing time increases linearly with the number of queries, because the average number of queries in each cell increases linearly as the number of queries increases. The performances of these two algorithms are shown in Fig.10.

F. *Effect of moving speed*

The speed of the moving object will affect the algorithm performance [19]. As the objects move faster, it is more likely that the results of the queries will be changed. Thus we can see the processing time will increase with higher speed in all these algorithms. It is also shown in Fig. 11. However, because OQ-Quad and CMP compute k-NN of moving query from scratch,

the performance of OQ-Quad and CMP is not influenced by the query speed.

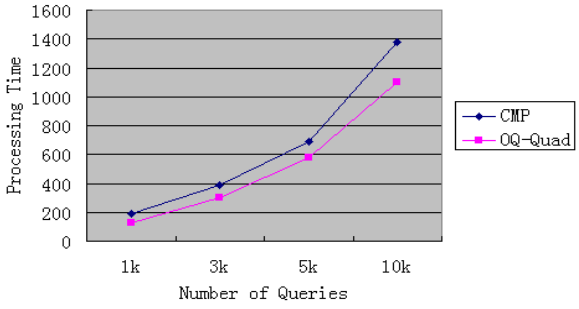


Figure 10. Overall processing time versus number of queries

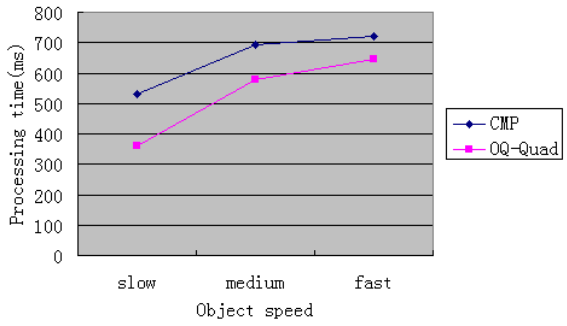


Figure 11. Overall processing time versus object speed

VI. CONCLUSIONS

We have studied the problem of CkNN query over moving objects. We present an object index and use level search method in the incremental query processing. The experiments have shown the benefits of our approach and demonstrated that OQ-Quad outperforms CMP in all settings.

REFERENCES

[1] Pfoser D, Jensen C S. Indexing of network constrained moving objects. In Proc. of 11st ACM Int. Symp. on Advances in Geographic Information Systems, New Orleans, Louisiana, USA, 2003, pp.25-32

[2] Saltenis S, Jensen C S. Indexing of Moving Objects for Location-Based Service. In Proc. of 18th Int. Conf. on Data Engineering, San Jose, CA,2002, pp.463-472.

[3] Song, ZX, and Roussopoulos, N, et al. K-Nearest Neighbor Search for Moving Query Point, In: Jensen CS, Schneider M, Seeger B, Tsotras VJ, eds. Proc. of the 7th Int'l Symp. on Spatial and Temporal Databases. Berlin: Springer-Verlag, 2001, pp79−96

[4] J Feng, T Watanabe. A fast method for continuous nearest target objects query on road network. Proc. of VSMM'02, 2002, pp182-191.

[5] Norio Katayama and Shinichi Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In Proc. ACM SIGMOD International Conference on Management of Data, pp. 369-380, May 1997.

[6] Apostolos Papadopoulos and Yannis Manolopoulos. In: Proceedings of ICDT, Delphi, Greece, 1997, pp394-408.

[7] R Benetis, C Jensen, et al. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In IDEAS, Los Alamitos, CA: IEEE Computer Society Press, 2002.

[8] Simonas Saltenis and Christian S. Jensen. Indexing of Moving Objects for Location-Based Services. In: Proc of Data Engineering,2002, pp463-472.

[9] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the Positions of Continuously Moving Objects. In: Proc.of SIGMOD'00, 2000, pp331-342.

[10] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and Querying Moving Objects. In: Proc, of the 1997 Intl. Conf. on Data Engineering, 1997,pp422-432.

[11] Xiong XP, Mokbel MF, Aref WG. SEA-CNN: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In: Proc. of the 21st ICDE Int'l Conf. on Data Engineering. IEEE ComputerSociety, 2005,pp643-654.

[12] Mouratidis K, Hadjieleftheriou M, Papadias D. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In: Özcan F, ed. Proc. of the 2005 SIGMOD Int'l Conf. on Management of Data. ACM Press, 2005, pp634-645.

[13] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao. A threshold-based algorithm for continuous monitoring of k nearest neighbors. IEEE TKDE,17(11),2005,pp1451–1464.

[14] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest and reverse nearest neighbor queries for moving objects. VLDB Journal,15(3),2006,pp229–250.

[15] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In ICDE, 2005, pp631-642.

[16] Frentzos, E., Gratsias, K., Pelekis, N, Nearest Neighbor Search on Moving ObjectTrajectories, [J].GeoInformatica,11(2), 2007, pp159-193.

[17] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In SIGMOD, 2006, pp635–646.

[18] Brinkhoff, T. A Framework for Generating Networkbased Moving Objects. GeoInformatica, (6)2,2002, pp153-180.

[19] Kun-Lung Wu; Shyh-Kwei Chen; Yu, P.S. Incremental Processing of Continual Range Queries over Moving Objects. In IEEE Transactions on Knowledge and Data Engineering, 18(11), 2006, pp1560-1575

 978-1-4244-3521-0/09/$25.00 ©2009 IEEE