Efficient Processing of Past-Future Spatiotemporal Queries

Katerina Raptopoulou
Department of Informatics
Aristotle University
of Thessaloniki
541 24 Greece
katerina@delab.csd.auth.gr

Michael Vassilakopoulos
Department of Informatics
Technological Educational
Institute of Thessaloniki
P.O. 141, 574 00 Greece
vasilako@it.teithe.gr

Yannis Manolopoulos
Department of Informatics
Aristotle University
of Thessaloniki
541 24 Greece
manolopo@delab.csd.auth.gr

ABSTRACT

Spatiotemporal databases emerge as an evolving scientific field due to a great variety of applications, tracking mobile objects being one of them. For this purpose, a number of methods have been proposed to efficiently organize and index moving objects and answer spatiotemporal queries. The majority of all these methods are addressing either the past or the future movement of the moving objects. Up until now, addressing both the past and the future movement of the objects in an integrated manner has rarely appeared in the literature. In the current paper, based on a spatiotemporal access method, the XBR-tree, we propose algorithms for the efficient processing of spatiotemporal window (past) and timestamp (past, present and future) queries. Moreover, we experimentally study the efficiency of processing these queries based on the XBR-tree against using an existing structure, the R^{PPF}-tree.

Categories and Subject Descriptors

H.2.2 [**Database Management**]: Physical Design—*Access methods*; H.2.4 [**Database Management**]: Systems—*Query processing*; H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Moving Objects, Spatiotemporal Databases / Queries

1. INTRODUCTION

During the past few years, several advances in mobile computing and wireless technologies have taken place. Hand held devices, mobile phones and wireless Internet terminals are continuously increasing in number. Tracking these mobile objects constitutes an important issue and will become critical in the years to come.

Additionally, there are several modern applications which include vehicle navigation, tracking and monitoring. In these applications, the positions of air, sea or land-based objects, such as airplanes, fishing boats and cars are of interest. An example of such applications, is to keep track of fighter planes which move very fast in air-force combat situations. Other real life examples that involve the use of objects with changing positions over time are: traffic control and fleet management. Geographical Information Systems are also a source of spatiotemporal data and require support by spatiotemporal database techniques. Moreover, the problem of video (or multimedia, in general) database management is relevant: objects appearing in each frame can be considered as two-dimensional moving objects, which one may want to keep track of.

All the abovementioned categories of applications have led to the proposal of a large number of methods to efficiently handle the organization, indexing and querying of moving objects. These methods can be classified in three categories, according to the time period of the movement of the objects: the past, future, or past-future time period.

As far as spatiotemporal queries are concerned, a variety of types have been proposed. Nevertheless, the types that seem to have become dominant are the following:

- **window query:** we are given a rectangle R and a time interval $[t_s, t_e]$ and we return all the moving objects, that intersect it from t_s to t_e.
- **nearest neighbor query:** we are given a moving object o and a time interval $[t_s, t_e]$ and we determine the nearest neighbors of o from t_s to t_e.
- **join query:** we are given two moving datasets S_1 and S_2 and a time interval $[t_s, t_e]$ and we calculate all the pairs (s_1, s_2), with $s_1 \in S_1$ and $s_2 \in S_2$ such that s_1 and s_2 meet, or are, at most, at a specified distance apart, at some time point from t_s to t_e.

In the current paper, we aim at handling both the past and the future movement of the moving objects. For this purpose, we extend a historical indexing method, namely the XBR-tree, to deal with the future movement as well. Additionally, the query types on which we focus are both window and timestamp . The window query is related to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'06 April 23-27, 2006, Dijon, France
Copyright 2006 ACM 1-59593-108-2/06/0004 ...$5.00.

a past time range, while the timestamp query is related to the past, present or future. We experimentally compare the I/O performance of this technique with the performance of a previously proposed technique, the R^{PPF}-tree.

The rest of the paper is organized as follows: Section 2 presents previous works that refer to the handling of moving objects. Section 3 describes in detail the assumptions under which the technique that we propose works. The R^{PPF}-tree is described in Section 4, while Section 5 presents the XBR-tree. Section 6 presents the experimental environment and results. Finally, in Section 7 we present our conclusions and our plans for future work in this topic.

2. RELATED WORK

As far as the theoretical background is concerned, Sistla et al. [10] proposed a data model called Moving Objects Spatiotemporal (MOST) data model that is capable to represent moving objects. The authors also proposed a query language called Future Temporal Logic. Moreover, Wolfson et al. addressed issues related to uncertainty and the frequency of updating the locations of the moving objects in the database [15].

In the literature, a large number of indexing schemes to efficiently organize moving objects and answer spatiotemporal queries have been proposed. The majority of these schemes refer to the past movement of the objects. Some of these indexing methods are described in the following.

Pfoser et al. suggested the STR-tree, an R-tree based indexing scheme suitable for the past movement of the moving objects [7]. The authors also examined trajectory-based queries. Furthermore, Hilbert R-trees were proposed by Nascimento et al. as an indexing method for spatiotemporal data and range queries [5]. Zhu et al. proposed octagon trees (OT-trees, O-trees), a structure able to index moving objects and handle range queries [16].

In [13], a Quadtree based indexing method for spatiotemporal data to handle range queries was proposed. In [8, 14], another quadtree based indexing method called the XBR-tree, where the movement of the objects is related to history, has been proposed. Furthermore, the processing of window queries based on XBR-trees has been presented. In the current paper, we further extend the XBR-tree based techniques to cover the future movement of the moving objects as well.

Additionally, Hadjieleftheriou et al. proposed the Partially Persistent (PPR-tree) as a method for indexing and querying the history of the moving objects [3]. In their implementation, the authors considered that the objects have a changing extent (for example, a moving object could shrink). Furthermore, the movement of the objects was described by polynomial functions (and not by linear ones) and the implemented queries were range ones.

Apart from the indexing methods for the past movement of the objects, in the literature there exists a limited number of research efforts that is related to the support of the future movement of the objects. For example, Saltenis et al. proposed an R^*-tree based indexing method (TPR-tree) to index the current and the future locations of the moving objects [9]. The authors have implemented algorithms to handle range queries as well. Another indexing method was suggested by Agarwal et al. in [1], who extended the External Range Tree to form the Kinetic Range Tree. This indexing scheme is capable to index the future locations of the moving objects and support range queries.

Other attempts have also been made under a different approach, namely, the use of transformations to index the trajectories of the moving objects. In [4], Kollios et al. used the dual transformation to improve the performance during range queries. Similarly, Chon et al. [2]. proposed the SVmodel as an alternative method of transformation.

Currently, there exits a very limited number of indexing methods able to handle both the past and the future movement of the moving objects. Sun et al [11] proposed an Adaptive Multi-Dimensional Histogram (AMH), which in each bucket stores the rectangular extend R, the average frequency f of all the cells in R and the average of the squared frequency g of those cells. The implemented queries were aggregate ones. In such a query $q(q_R, q_T)$, the result returns the number of objects which fall in a rectangle region q_R at timestamp q_T. If $q_T = 0$ then the aggregation query is a present timestamp, if $q_T < 0$ then the aggregation query has a historical timestamp, whereas if $q_T > 0$ then the aggregation query has a future timestamp. In [6] Pelanis et al. proposed an index (R^{PPF}-tree) to support both the past and the future movement of the objects. The implemented query types are only timestamp ones, for all points in time.

In this paper, we extend the XBR-tree, a historical tree, to deal with future, as well. Moreover, we implement algorithms to handle both window and timestamp queries. Finally, we compare the XBR-tree with the R^{PPF}-tree, for both of these query types.

3. ASSUMPTIONS

We made several assumptions with respect to the movement of the objects. In the sequel, we describe how the objects are represented and how the updates are carried out.

We assume that time is discrete and that the location and the velocity vector (direction of movement and speed) of each object is updated only at predefined time points that divide time in a number of time intervals. For each time interval of the past (up to the current time point), a line segment that expresses the movement of each object during this interval is maintained. For the interval starting at the current time point, a line segment that expresses the initial location and velocity vector of each object is maintained. All these line segments make up a polyline that expresses the trajectory of each object from the starting time point to the point that follows the current time point. Especially, the last line segment expresses not the actual trajectory, but the expected trajectory from the current time point to the next one.

When time advances to the next time point, each object notifies the system of its actual location and velocity vector. With this data, the last line segment of the polyline is updated (meaning that, in general, the last line segment must be deleted and reinserted to reflect the actual data) and a new segment that expresses the expected trajectory from the new current time point to the next one is inserted. Note that the use of the last line segment of the trajectory of each object that corresponds to the future allows to answer predictive queries about the positions of the moving objects in the near future.

Although, it is possible to handle both the x and the y coordinate of each object (along with time) at the same structure (with tree versions that can handle 3-dimensional data), following the approach of [13], we handle x and y coor-

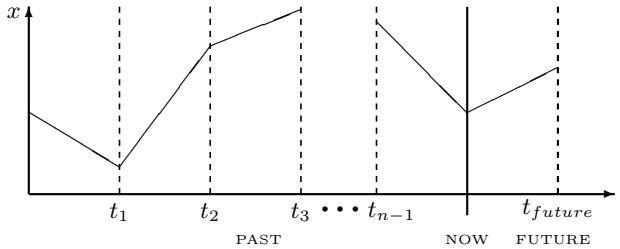

Figure 1: Monitoring of Moving Objects

dinates independently with XBR-trees (unlike R^{PPF}-trees). This means that we keep one 2-dimensional tree for the X coordinate, along with time and another 2-dimensional tree for the Y coordinate, along with time. We answer a query using each of the trees and then combine the subanswers. Accordingly, at each time point, we update both trees. In Figure 1, the movement of an object along the x-axis (vertical axis) in the course of time (horizontal axis) is depicted.

4. THE R^{PPF}-TREE

The R^{PPF}-tree [6] consists of a partially persistent R-tree used to store the past and the present trajectories of the moving objects and a partially persistent TPR-tree used to store the present and the future trajectories. More details about the partially persistent R-tree appear in [12], whereas the partially persistent TPR-tree is briefly described in the sequel.

4.1 TPR Internal and External Nodes

The internal nodes are of the form: $< ptr, tpbr, t-, -t >$, where *ptr* is a pointer to a child node, *tpbr* is a Time-Parameterized Bounding Rectangle, and $[t-, -t]$ is the interval of time. The external nodes are of the form: $< oid, tpp, t-, -t >$, where *oid* is the object identifier, *tpp* is a time-parameterized point and $[t-, -t]$ is the interval of the validity. $tpp = (x; u) = [x_1, x_2, \ldots, x_m; u_1, u_2, \ldots, u_m]$ with x_i and u_i the coordinates of the position and the velocity of the object at time $t-$.

The Time-Parameterized Bounding Rectangles (TPBRs) of the TPR-tree are not necessarily valid for the past time points. Moreover, they do not take into consideration the different insertion and deletion times of the moving objects. Therefore, they cannot be used in the partially persistent TPR-tree. In the next section, we refer to different TPBR variations that deal with this problem.

4.2 Optimized and Double TPBRs

When computing an optimized TPBR in two dimensions, its spatial and velocity coordinates should be chosen to minimize the integral of the area of the bounding rectangle, from $t-$ to $CT + H$ (CT represents current time, while H represents H time units in the future and it is a workload-specific parameter). In more dimensions, a near-optimal TPBR can be computed by combining the solutions of the different dimensions. The disadvantage of the optimized TPBRs is that their computation is complex and that they cannot be tightened. Therefore, in our implementation we chose the double TPBRs.

The double TPBRs are TPBRs which allow tightening. Each TPBR is divided into two parts the "head" and the "tail". The tail begins at the time of the last update (insertion or deletion), t_{lu}, and extends to the infinity. The tail is a common TPBR of the TPR-tree. The head bounds the finite segments of the trajectories, from $t-$ to t_{lu}. This can be done by using either an optimized TPBR or a common TPBR of the TPR-tree.

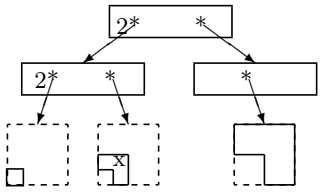

Figure 2: An XBR-tree

5. THE XBR-TREE

The XBR-tree [8] consists of two node types. The first type is for the internal nodes that form a multiway index. The second node type is for the leaves which store the line segments that are inserted in the tree. Both node types are being kept on disk. Due to space limitations and for the sake of presentation, we assume 2 dimensions. For 2 dimensions the hierarchical decomposition of the space is the same as the one in quadtrees. More specifically, the space is subdivided into 4 equal subquadrants, any of which may be further recursively subdivided into 4 subquadrants.

5.1 Internal Nodes

In an internal node, pairs of the form <address, pointer> are contained. The number of these pairs is non-predefined because the addresses being used are of variable size. An address expresses a child node region and is paired with the pointer to this child node. Apparently, both the size of an address and the total space occupied by all pairs within a node must not exceed the node size. The addresses in these pairs are used to represent certain subquadrants that result from the repetitive subdivision of the initial space. This is done by assigning the numbers 0, 1, 2 and 3 to NW, NE, SW and SW quadrants respectively. For example the address 1 is used to represent the NE quadrant of the initial space, while the address 10 to represent the NW subquadrant of the NE quadrant of the initial space. The XBR-tree, introduces a new idea. That is, the region of a child is the subquadrant specified by the address in its pair, minus the subquadrants corresponding to all the previous pairs of the internal node to which it belongs.

Figure 2 depicts a three-level XBR-tree. The "*" being used in the figure denotes the end of each address. As shown in the figure, the left child of the internal node is the SW of the initial space, as it is denoted from its address 2*. Moreover, the right child of the internal node is the initial space if we subtract from it its SW subquadrant.

When a search or an insertion of a point is performed, descending the tree from the root specifies the appropriate leaves and their regions. At the root, the region that has to be checked is the whole space. When visiting an internal node, we check in turn every contained pair. The first pair with a subquadrant that contains the particular coordinates is chosen and its pointer to the next level is followed. By examining this way the pairs in each node, the region under consideration is refined, since we subtract the subquadrants of the pairs appearing to the left of this pair. The insertion,

or search procedure of a line segment in the XBR-tree is similar to the ones described above, with one key difference: a line segment is stored in the XBR-tree to all the leafs that it crosses. This means that during our descend from the root to the leaf nodes level, in each internal node, we sequentially examine the <address, pointer> pairs and recursively visit every (and not simply the first) child node with a region that is crossed by the specific line segment.

Whenever an internal node overflows, then a split into two occurs. This split is done in such a way, so that a good balance between the regions of the two resulting nodes is achieved. More details regarding internal node overflows can be found in [8].

5.2 Leaf Nodes

The leaves of the XBR-tree contain all the line segments inserted in the tree. The total number of line segments in each leaf node are restricted by a predefined capacity C which cannot be exceeded. When after an insertion of a line segment a specific leaf node overflows then it is split into four equal subquadrants.

All the resulting subquadrants that contain any of the lines segments of the old leaf node, are inserted in the internal node. The subquadrants that contain more line segments than the predefined capacity, store these segments in overflow pages. For example, if repetitive insertions occur in the SW subquadrant of the left child in the tree of the Figure 2, then this leaf will split into four subquadrants. Each and every resulting subquadrant, which includes any of the line segments, will be added to the internal node.

Deletion is used while updating the location and the velocity vector of each object, at each time point. That is, the last line segment of the trajectory of each moving object is updated at the end of each time interval (in general, it must be deleted and reinserted to reflect the actual data). In addition to the updated old line segment, a new one is inserted to express the expected trajectory from the new current to the next time point. Since a line segment may cross the regions of several XBR-tree leaf nodes, it has to be removed from all these leaf nodes. More details regarding deletions and the XBR-tree, in general, can be found in [8].

6. EXPERIMENTATION

For experimentation, we used a Pentium on 1,6 GHz and 1 GB memory. Furthermore, the page size was 4K, and, as a result, the number of the line segments in the leaves was 204. After thorough experimentation, we came to the conclusion, that a buffer of 100 K, with LRU shows improved performance in comparison with other methods, and therefore it has been adopted for the experiments execution.

The past time horizon was of 1000 distinct time units which were separated into 10 equal intervals, each one of 100 time units. The queries executed were of four types: past window queries and past, present and future timestamp queries. As a performance measure, we counted the number of the disc accesses for all four types of the queries. In all the experiments, the number of the moving objects N varied from 1000 to 10000. Additionally, the range of the queries was the 0.1, 0.01 and 0.001 fraction of the dataspace.

Due to space limitations, in this paper we graphically present an indicative part only of the results of our experiments. However, the trends remained the same for all the experiments performed. In left and center parts of Figure 3, we present the disc accesses for past window queries (the range of the queries equals 0.1 on the left part and 0.001 on the center part). In the right part of the same figure we present the number of disc accesses for past timestamp queries with range 0.01. The left part of Figure 4 depicts the number of disk accesses for present timestamp queries, where the query range equals 0.01. The center and right parts of the same figure, depict the number of disk accesses for future timestamp queries with range 0.1 and 0.001, respectively.

In all cases, the XBR-tree outperformed the R^{PPF}-tree. It is apparent from the above figures that the number of disk accesses of the XBR-tree is smaller than the number of disk accesses of R^{PPF}-tree, for all the queries executed. The superiority of XBR-tree can be explained by its compactness (it is a very compact tree, due to the compressed representation of addresses) and by the fact that there is no spatial overlapping between nodes and thus, no revisiting of the same nodes during queries execution.

7. CONCLUSIONS AND FUTURE WORK

The efficient organization, indexing and querying of moving objects is of great demand in several modern applications. In the current paper, we have adapted a historical tree, namely the XBR-tree, to handle the future movement of the objects. Moreover, we have implemented three types of timestamp queries: the past, the present and the future timestamp query. We have also implemented a past window query type. Our experiments show that the XBR-tree outperforms the only known thus far three structure for past and future queries, the R^{PPF}-tree.

As possible future work we consider the following:

- The implementation of other types of queries (apart from window and timestamp queries), such as join queries and nearest neighbor queries.
- The use of a single 3-dimensional XBR-tree (x, y, t) and not two 2-dimensional XBR-trees, one for the dimensions (x, t) and one for the dimensions (y, t).
- A study of the computational overheads required to deal with compression and uncompression of addresses, especially when calculating compliments of regions.

8. ACKNOWLEDGMENTS

Financially supported by the ARCHIMEDES project 2.2.14, "Management of Moving Objects and the WWW", of the Technological Educational Institute of Thessaloniki (EPEAEK II - funding by the Greek Ministry of Education and Religious Affairs and the European Union) and the Bilateral Greek-Serbian Scientific Protocol 2005-2006 (funding by the General Secretariat of Research and Technology).

9. REFERENCES

[1] Agarwal P.K., Arge L. and Erickson J.: "Indexing Moving Points", *Proc. ACM Symposium on Principles on Database Systems (PODS)*, pp.175-186, 2000.

[2] Chon H.D., Agarwal D. and Abbadi A.E.: "Storage and Retrieval of Moving Objects", *Proc. 2nd International Conference on Mobile Data Management (MDM)*, pp.173-184, 2001.

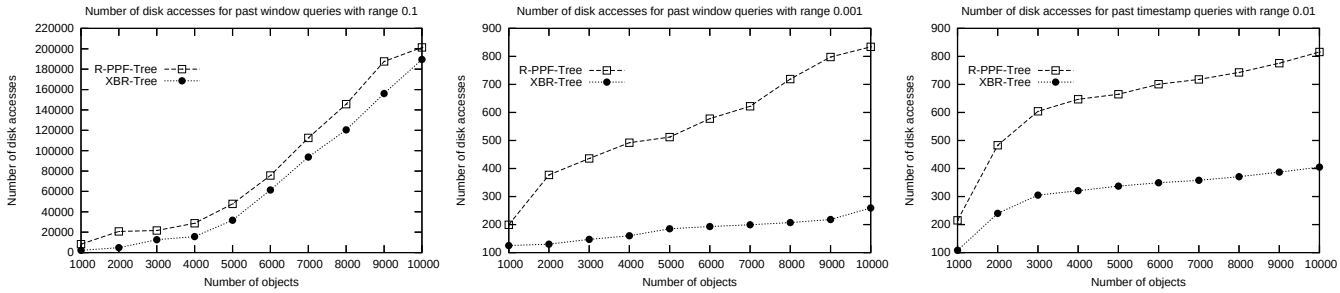

Figure 3: Number of disc accesses for past window queries with range 0.1 (left) and 0.001 (center) and for past timestamp queries with range 0.01(right).

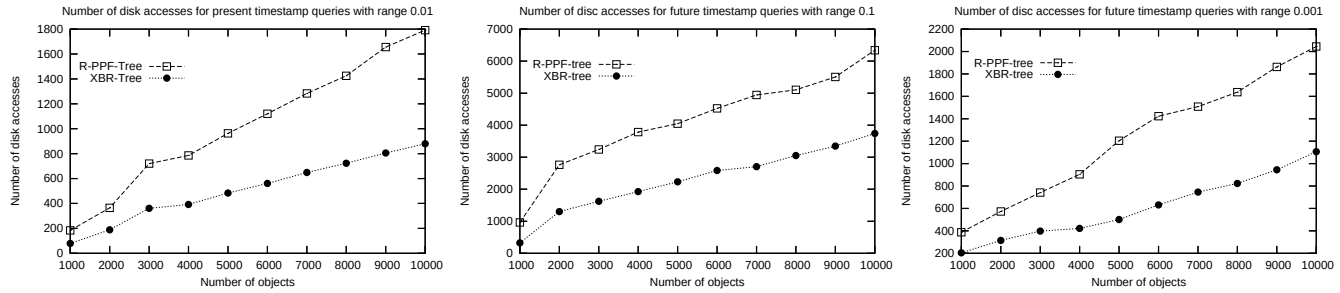

Figure 4: Number of disc accesses for present timestamp queries with range 0.01 (left) and for future timestamp queries with range 0.1 (center) and 0.001 (right).

[3] Hadjieleftheriou M., Kollios G., Tsotras V.J. and Gunopoulos D.: "Efficient Indexing of Spatiotemporal Objects", *Proc. 8th International Conference on Extending Database Technology (EDTB)*, pp.251-268, 2002.

[4] Kollios G., Gunopoulos D. and Tsotras V.: "On Indexing Mobile Objects", *Proc. ACM Symposium on Principles on Database Systems (PODS)*, pp.261-272, 1999.

[5] Nascimento M.A. and Silva J.R.O.: "Towards Historical R-trees", *Proc. ACM Symposium on Applied Computing (SAC)*, 1998.

[6] Pelanis M., Saltenis S. and Jensen C.S.: "Indexing the Past, Present and Anticipated Future Positions of Moving Objects", Tech. Report 78, Timecenter, 2004.

[7] Pfoser D., Jensen C.S. and Theodoridis Y.: "Novel Approaches to the Indexing of Moving Object Trajectories", *Proc. 26th International Conference on Very Large Databases (VLDB)*, pp.395-406, 2000.

[8] Raptopoulou K., Vassilakopoulos M. and Manolopoulos Y.: "Towards Quadtree-based Moving Objects Databases", *Proc. East-European Conference on Advances in Database Systems and Information Systems (ADBIS)*, 2004.

[9] Saltenis S., Jensen C.S., Leutenegger S. and Lopez M.: "Indexing the Positions of Continuously Moving Objects", *Proc. ACM Conference on Management of Data (SIGMOD)*, pp.331-342, 2000.

[10] Sistla A.P., Wolfson O., Chamberlain S. and Dao S.: "Modeling and Querying Moving Objects", *Proc. 13th IEEE International Conference on Data Engineering (ICDE)*, pp.422-432, 1997.

[11] Sun J., Papadias D., Tao Y. and Liu B.: "Querying about the Past, the Present and the Future in Spatiotemporal Databases", *Proc. 20th International Conference on Data Engineering (ICDE)*, pp.202-213, 2004.

[12] Tao Y. and Papadias D.: "MV3R-tree - a Spatiotemporal Access Method for Timestamp and Interval Queries", *Proc. 27th International Conference on Very Large Databases (VLDB)*, pp.431-440, 2001.

[13] Tayeb J., Ulusoy O. and Wolfson O.: "A Quadtree based Dynamic Attribute Indexing Method", *The Computer Journal*, Vol.41, No.3, pp.185-200, 1998.

[14] Vassilakopoulos M. and Manolopoulos Y.: "External Balanced Regular (x-BR) Trees: new Structures for Very Large Spatial Databases", *Proceeding 7th Panhellenic Conference on Informatics*, pp.III.61-III.68, 1999.

[15] Wolfson O., Xu B., Chamberlain S. and Jiang L.: "Moving Objects Databases: Issues and Solutions", *Proc. International Conference on Statistical and Scientific Database Management (SSDBM)*, pp.111-122, 1998.

[16] Zhu H., Su J. and Ibarra O.H.: "Trajectory Queries and Octagons in Moving Object Databases", *Proc. 11th ACM International Conference on Information and Knowledge Management (CIKM)*, pp.413-421, McLean, VA, 2002.