Indexing the Trajectories of Moving Objects in Networks
(Extended Abstract)

Victor Teixeira de Almeida Ralf Hartmut Güting
Praktische Informatik IV
Fernuniversität Hagen, D-58084 Hagen, Germany
{victor.almeida, rhg}@fernuni-hagen.de

Abstract

The management of moving objects has been intensively studied in recent years. A wide and increasing range of database applications has to deal with spatial objects whose position changes continuously over time. The main interest of these applications is to efficiently store and query the positions of these objects. To achieve this goal, index structures are required. Most of the proposals of index structures for moving objects deal with unconstrained 2-dimensional movement. The constrained movement is a special and a very important case of object movement. In this paper we propose a new index structure for moving objects in networks, the MON-Tree. We tested our proposal in an experimental evaluation with generated data sets. The MON-Tree showed good scalability when increasing the number of objects and time units in the index structure, and the query window and time interval in querying.

1. Introduction

The management of moving objects has been intensively studied in recent years. A wide and increasing range of database applications has to deal with spatial objects whose position changes over time, such as cars, trains, air planes, and many more examples. The main interest of these applications is to store and efficiently query the positions of continuously moving objects.

Indexing techniques have been used since the advent of relational database management systems with success. Indexing is even more important when the data is more complex and, for spatial databases systems, due to high performance requirements, access methods should be used on every relation for supporting queries efficiently. Following these ideas, for moving objects databases, which is a spatio-temporal application, and consequently more complex, the need of good indexing techniques is even more important.

There is a large number of proposals in the literature to index the trajectories of moving objects, e.g., [3, 4, 8, 11–14]. These works assume free movement of the objects in the 2-dimensional space. Sometimes, the movement of the objects is constrained to networks, e.g. cars and trains. The indexing approaches can take advantage of this knowledge. Recently, two index structures for indexing the trajectories of moving objects in networks have been proposed in [5, 9]. Both use the same idea of converting a 3-dimensional problem into two sub-problems with lower dimensions. The main disadvantage of these approaches is the model of network used, where each edge in the network can represent only a single line segment. This model leads to a high number of entries and lots of updates in the index structure, because distinct entries are needed for every line segment the object traverses.

In this paper we propose a new index structure, the **M**oving **O**bjects in **N**etworks **Tree** (MON-Tree) to efficiently store and retrieve the trajectories of objects moving in networks. We use two different network models that can be indexed by the MON-Tree. The first model is edge oriented, i.e., the network is composed by edges and nodes and each edge has an associated polyline. The second one is route oriented, i.e., the network is composed by routes and a set of junctions between these routes. We test our proposal in an experimental evaluation with generated data sets using as underlying networks the roads and railroads of Germany.

The structure of this paper is as follows: Section 3 proposes the MON-Tree index structure and the insert and search algorithms. Section 4 experimentally evaluates our proposed index structure, and finally, Section 5 concludes the paper. A full version of this paper can be found in [1].

2. The Network Models

In this section we describe in more detail the two different network models that can be indexed by the MON-Tree.

In the first model, a network is a graph $G = (N, E)$ where N is a set of nodes and $E = N \times N$ is a set of edges. A node $n \in N$ has an associated point $p_n = (x, y)$ in the 2-dimensional space and an edge $e \in E$ connects two nodes n_{1e} and n_{2e} and has an associated polyline $l_e = p_1, \ldots, p_k$, where p_i are 2-dimensional points, $1 \leq i \leq k$, k is the size of the edge, $p_1 = p_{n_1}$, and $p_k = p_{n_2}$. A position *epos* inside an edge e is represented by a real number between 0 and 1, where 0 means that the position lies on the node n_{1e} and 1 means that the position lies on the node n_{2e} of the edge. The domain of a moving object position inside a graph G is $D(G) = E \times pos$. The time is given by a time domain T isomorphic to real numbers. A moving object then, is a partial function $f : T \rightarrow D(G)$. As an example of the usage of this model, we can cite [10].

This first model is simple and straightforward, but not the best one to represent transportation networks. In [6] we extended the framework in [7] to handle network constrained movement, where a route oriented model is used. In this model, the network is represented in terms of routes and junctions between the routes, i.e., a network $G' = (R, J)$, where R is a set of routes and J is a set of junctions. A route $r \in R$ has an associated polyline $l_r = p_1, \ldots, p_k$, where p_i are 2-dimensional points, $1 \leq i \leq k$, and k is the size of the route. A position *rpos* inside a route r is represented by a real number between 0 and 1, where 0 means that the position lies on the point p_1 and 1 means that the position lies on the point p_k of the route. A junction $j \in J$ is represented by two routes r_1 and r_2 and two route positions $rpos_{r_1}$ and $rpos_{r_2}$. The domain of a moving object position inside a graph G' is $D'(G') = R \times rpos$. The time domain T is the same and then, a moving object in this second model is a partial function $f : T \rightarrow D'(G')$.

A detailed discussion about why using the route oriented model is given in [6], but the most practical reason to use the route oriented model instead of the edge oriented one, taking indexing techniques into consideration, is that the representation of a moving object becomes much smaller in this way.

3. The MON-Tree

In this section we propose a new index structure to efficiently store and retrieve past states of objects moving in networks, the MON-Tree. The Section 3.1 presents the index structure, and the insertion and search algorithms are presented in Section 3.2 and 3.3, respectively.

3.1. Index Structure

The index structure proposed in this paper assumes that objects move along polylines, which can belong to edges, for the first network model, or to routes, for the second network model. The index structure is composed by a 2D R-Tree (the *top R-Tree*) indexing polyline bounding boxes and a set of 2D R-Trees (the *bottom R-Trees*) indexing objects' movements along the polylines.

We also use a hash structure in the top level containing entries of the form $\langle polyid, bottreept \rangle$, where *polyid* is the polyline identification and *bottreept* is a pointer to the corresponding bottom R-Tree. The hash structure is organized by *polyid*.

Hence, we have two top level index structures: an R-Tree and a hash structure; pointing to bottom level R-Trees. The need for these two top level index structures is as follows: on the one hand, the insertion algorithm of moving objects takes a polyline identification as an argument, and then uses the top level hash structure to find the bottom level R-Tree into which the movement should be inserted. On the other hand, the search algorithm takes a spatio-temporal window as an argument and starts the search on the top R-Tree, which contains the polylines' bounding boxes.

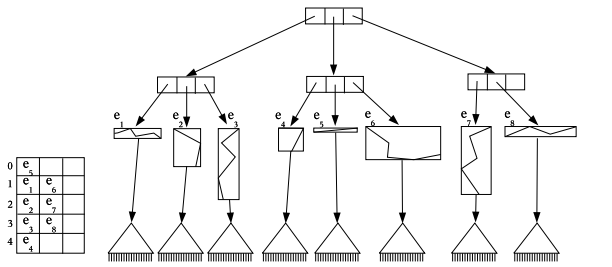

Figure 1. Example of the MON-Tree index structure.

An example of the MON-Tree index structure can be seen in Figure 1. In the top R-Tree, the polylines are indexed using a MBB approximation. In this way, the leaves of this tree contain the information $\langle mbb, polypt, treept \rangle$, where *mbb* is the MBB of the polyline, *polypt* points to the real representation of the polyline, and *treept* points to the corresponding bottom R-Tree of that polyline. Internal nodes have the following information $\langle mbb, childpt \rangle$, where *mbb* is the MBB that

contains all MBBs of the entries in the child node, and *childpt* is a pointer to the child node.

The bottom R-Tree indexes the movement of the objects inside a polyline. The movement is represented by the position interval (p_1, p_2) and a time interval (t_1, t_2), where $0 \leq p_1, p_2 \leq 1$. These two values p_1 and p_2 store the relative position of the objects inside the polyline at times t_1 and t_2 respectively.

3.2. Insertion

In this index structure we allow two different kinds of insertion: polyline insertion and movement insertion. A polyline insertion is needed to construct the basis network. The moving object insertion is necessary every time an object is created or it changes its motion vector, i.e., its speed and/or direction. It is also necessary to perform a moving object insertion every time an object changes from one polyline to another.

Polyline Insertion. The algorithm for polyline insertion is very simple: just insert the polyline identification with a *null* pointer in the hash structure. The insertion of the polyline in the top R-Tree is postponed to the insertion of the first moving object traversing it. The reason for this approach is to avoid having polylines without moving objects in the top R-Tree, while they do not participate in queries. In this way, we keep the top R-Tree as small as possible.

Movement Insertion. The movement insertion algorithm takes as arguments the moving object identification *moid*, the polyline identification *polyid*, the position interval $p = (p_1, p_2)$ where the object moved along the polyline, and the corresponding movement time interval $t = (t_1, t_2)$. The algorithm starts looking in the top hash structure to find the associated polyline, i.e., the polyline which has identification number equal to *polyid*. If the polyline does not have an associated bottom R-Tree yet, then a new one is created and the polyline's MBB is inserted on the top R-Tree. The pointer to this newly created bottom R-Tree is updated in the top hash structure. Now, given the associated bottom R-Tree, the rectangle (p_1, p_2, t_1, t_2) is inserted into it using the insert algorithm of the R-Tree.

3.3. Search

Given a spatio-temporal query window $w = (x_1, x_2, y_1, y_2, t_1, t_2)$, the query of the form: "find all objects that have lied within the area $r = (x_1, x_2, y_1, y_2)$, during the time interval $t = (t_1, t_2)$" are expected to be the most common ones addressed by spatio-temporal database management system users. This query is commonly called *range query* in the literature. A variant of this query is to find only the pieces of the objects' movements that intersect the query window w. We call this a *window query*. The main functionality of the MON-Tree index is to answer these two kinds of query.

For the window query, the algorithm receives a spatio-temporal query window w and proceeds in three steps. In the first step, a search in the top R-Tree is performed to find the polylines' MBBs that intersect the query spatial window r. Then, in the second step, the intervals where the polyline intersects the spatial query window r are found using the real polyline representation. It is important to note that this procedure is done in main memory and the result is a set of windows $w' = \{(p_{11}, p_{12}, t_1, t_2), \ldots, (p_{n1}, p_{n2}, t_1, t_2)\}$, where n is the set size, $n \geq 1$, and the interval (t_1, t_2) is the query time interval t. Moreover, the windows are disjoint and ordered, i.e., $p_{i1} \leq p_{i2} \wedge p_{i2} < p_{(i+1)1}$, $1 \leq i \leq n-1$. An example of the result of this procedure can be seen in Figure 2.

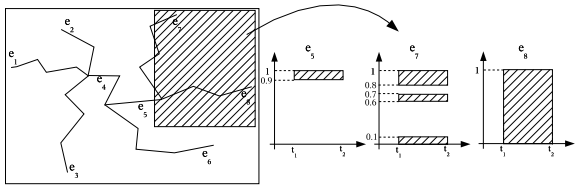

Figure 2. Example of the interval set determination in the search algorithm.

Given this set of windows w', in the third step, the bottom R-Trees are searched using a modified algorithm for searching a set of windows, instead of only one. The set of query windows w' is passed as an argument to the search algorithm and the R-Tree search algorithm is modified to handle multiple query windows (see [1] for details).

For the range query processing, we need an additional step after the third step of the window query to remove duplicates and return only the objects' identification. This step can be done in memory, i.e., the objects' identifications found in the third step of the window query can be stored in a main memory structure (an array for example) and after its completion, a duplicate removal is done.

4. Experimental Evaluation

In order to examine the performance of our proposed index structure, the MON-Tree, we did an experimental evaluation. We implemented the FNR-Tree

and compared to our results. In all our experiments, we used the network-based moving objects generator proposed in [2]. We used two networks, the roads and the railroads of Germany downloaded from the *Geo Community* web site[1].

In order to show the good scalability of the proposed index structure, we varied the number of objects, the number of time units, and the disk page size in the data sets and index construction. We also varied the size of the query window, the size of the query time interval, and the cache size in query processing. The complete set of experiments can be found in [1]. The MON-Tree clearly outperformed the FNR-Tree in all tests, and the MON-Tree indexing the route oriented model showed the best results.

Unfortunately, we slightly misunderstood the structure of the FNR-Tree in our experimental evaluation in [1], which was recently pointed out to us by its authors. Preliminary experiments showed that the structure we tested was somewhat less efficient than the real FNR-Tree described in [5], but that the MON-Tree still has a considerably better performance than the real FNR-Tree. We plan to redo all the experiments in the near future and to revise our paper [1] accordingly.

5. Conclusions

In this paper we proposed a new index structure for moving objects on networks, the MON-Tree. The MON-Tree stores the complete trajectories of the objects moving in networks. There are two network models that can be indexed by the MON-Tree: an edge oriented model and a route oriented one. The MON-Tree is capable of answering two kinds of query: the range query and the window query, both on past states of the data. We have experimentally evaluated our proposed index structure with generated data sets. The MON-Tree showed good scalability and outperformed the concurrent in all our tests.

Acknowledgments

This work was partially supported by a grant Gu 293/8-1 from the Deutsche Forschungsgemeinschaft (DFG), project "Datenbanken für bewegte Objecte" (Databases for Moving Objects). The authors would like to thank Prof. Dr. Thomas Brinkhoff for providing the network-based data generator and especially for providing some direct support.

1 `http://data.geocomm.com/catalog/GM/group103.html`

References

[1] V. T. Almeida and R. H. Güting. Indexing the trajectories of moving objects in networks. Technical Report 309, Fernuniversität Hagen, Fachbereich Informatik, 2004.

[2] T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.

[3] H. D. Chon, D. Agrawal, and A. E. Abbadi. Using space-time grid for efficient management of moving objects. In *2nd ACM Intl. Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*, pages 59–65, 2001.

[4] H. D. Chon, D. Agrawal, and A. E. Abbadi. Query processing for moving objects with space-time grid storage model. In *Proc. of the 3rd Intl. Conf. on Mobile Data Management (MDM)*, pages 121–, 2002.

[5] E. Frentzos. Indexing objects moving on fixed networks. In *Proc. of the 8th Intl. Symp. on Spatial and Temporal Databases (SSTD)*, pages 289–305, 2003.

[6] R. H. Güting, V. T. Almeida, and Z. Ding. Modeling and querying moving objects in networks. Technical Report 308, Fernuniversität Hagen, Fachbereich Informatik, 2004.

[7] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Transactions on Database Systems (TODS)*, 25(1):1–42, 2000.

[8] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and G. Gunopoulos. Efficient indexing of spatiotemporal objects. In *Proc. of the 8th Intl. Conf. on Extending Database Technology (EDBT)*, pages 251–268, 2002.

[9] C. S. Jensen and D. Pfoser. Indexing of network constrained moving objects. In *Proc. of the 11th Intl. Symp. on Advances in Geographic Information Systems (ACM-GIS)*, 2003.

[10] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proc. of 29th Intl. Conf. on Very Large Data Bases (VLDB)*, pages 802–813, 2003.

[11] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In *Proc. of Advances in Spatial Databases, 6th Intl. Symp. (SSD)*, pages 111–132, 1999.

[12] K. Portkaew, I. Lazaridis, and S. Mehrotra. Querying mobile objects in spatio-temporal databases. In *Proc. of the 7th Intl. Symp. on Spatial and Temporal Databases (SSTD)*, pages 59–78, 2001.

[13] Z. Song and N. Roussopoulos. Hashing moving objects. In *Proc. of the 2nd Intl. Conf. on Mobile Data Management (MDM)*, pages 161–172, 2001.

[14] Z. Song and N. Roussopoulos. SEB-tree: An approach to index continuously moving objects. In *Proc. of the 4th Intl. Conf. on Mobile Data Management (MDM)*, pages 340–344, 2003.