INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN 2278 – 0211 (Online)

A Survey on Spatial Indexing of Trajectories using Adaptive Network R-Tree of Moving Objects in Road Networks

S. Sathya
Research Scholar, Department of CS, Bharathiar University, Coimbatore, India
Dr. M. Sugumaran
Professor, Department of CSE, Pondicherry Engineering College, India

Abstract:
For querying large amounts of moving objects, a key problem is to create efficient indexing structures that make it possible to effectively answer various types of queries. Traditional spatial indexing approaches cannot be used because the locations of moving objects are highly dynamic, which go ahead to frequent updates of index structures, which in turn will cause huge expenditure. In earlier researches states that the functional approaches to manage moving objects in Euclidean spaces. In this paper, to describe the techniques that effectively indexing the positions of moving objects in spatial networks and supports the heavy update loads. Not only that and also increases the efficiency of the query extraction. The performance study, comparing this indexing method with FNR-Tree, MON-Tree, and TPR-Tree, shows that ANR-Tree outperforms them.

Key words: *spatial index, moving object database, Adaptive Network R-tree, direct access table*

1. Introduction

Developing efficient index structures is an important research issue for moving object database. Traditional spatial index structures are not appropriate for indexing moving objects because the constantly changing locations of objects requires constant updates to the index structures and thus greatly degrades their performance.

According to reference [1], applications dealing with moving objects can be grouped into three movement scenarios, namely unconstrained movement, constrained movement, and movement in transportation networks. Most works for indexing moving objects assume free movement of the objects in space. But in many applications, movement occurs on fixed road network. In this paper, we propose spatial indexing for the past, present, and anticipated future trajectories of moving objects using the ANR-tree (for dynamic updates). The amount of historical trajectories is constantly increasing over time, which makes it infeasible to keep track of all location updates. As a result, past positions of moving objects are often approximated by polylines (multiple line segments). Several indexing techniques [10, 11, 15], all based on 3-dimensional variations of R-trees [7] and R*-trees [4], have been proposed with the help of ANR-tree, and their goal is to minimize storage and query cost.

2. Common Problems in Spatial Indexing

- Some of the early works [2, 9] employ dual transformation techniques that represent predicted positions as points moving in a two-dimensional space.
- However, they are largely hypothetical, and applicable either only in 1D space [9] or entirely inapplicable in practice due to some large hidden constant in complexity.

3. Related Work

3.1. Indexing Moving Objects in Spatial Networks

Indexing methods for the moving objects focus on two issues: (1) storage and retrieval of historical information, and (2) prediction of future trajectory. Indexing future trajectories raises different problems when compared to indexing historical trajectories. The goal is to efficiently retrieve objects that will satisfy some spatial condition at a future time given their present motion vectors. The proposed methods are,

- To manage moving objects in spatially constrained networks, Pfoser etal. [12] proposed to convert the three-dimension problem into two sub-problems of lower dimensions through certain transformation of the networks and the trajectories.
- Another approach, known as the Fixed Network R-tree (FNR) [6], separates spatial and temporal components of the trajectories and indexes the time intervals during which any moving object was on a given network link.
- The Moving Object Network (MON-tree) approach [3] [4] further improves the performance of the FNR-tree by representing each edge by multiple line segments (i.e., polylines) instead of just one line segment.
- The main focus of the most recent studies is on practical implementation, for instance, the time-parameterized R-tree (TPR-tree) [14] and its variations[13, 16] are based on R-tree [7], and the *Bx*-tree [8] and its variations [17] are based on *B*+-tree. These structures use the linear prediction model to support predictive queries and to reduce the number of index updates. However, the assumption of linear movement limits their applicability in a majority of real-life applications especially in traffic networks where vehicles change their velocities frequently.

4. Solution Approach

This paper addresses the problem of efficient indexing of moving objects in spatial networks for supporting heavy loads of updates. We exploit the constraints of the network and the stochastic behavior of the real traffic to achieve both high update and query efficiency.

Introducing a dynamic data structure called Adaptive Unit (AU) to group neighboring objects with similar movement patterns in the network. A spatial index (e.g., R-tree) for the road network is then built over the adaptive units to form the index scheme (e.g., the adaptive network R-tree) for moving objects in road networks.

4.1. The Adaptive Unit

Conceptually, an adaptive unit is similar to a 1-dimensional Minimum Bounding Rectangles or MBR [5] in the TPR-tree, which expands with time according to the predicted movement of the objects, it contain.

However, in the TPR-tree, it is possible that an MBR may contain objects moving in opposite directions, or objects moving at different speeds. As a result, the MBR may expand rapidly, which may create large overlap with other MBRs. The adaptive unit avoids this problem by grouping objects having similar moving patterns. Specifically, for objects in the same edge of the spatial network, use a distance threshold and a speed threshold to cluster the adjacent objects with the same direction and similar speed. Figure 1 illustrates the difference between an AU and an MBR in the TPR-tree [14].

Utilize an arrow to symbolize the direction and the speed of an object. In Fig. 4.1(b), objects *e* and *f* are not in the AU because they have different directions and belong to different cluster. As shown in Fig. 4.1(c), the MBR in the TPR-tree experiences significant expanding as time goes by. In comparison, as Fig. 4.1(d) shows, the AU has no obvious enlargement because objects in the AU move in a cluster. Thus, AUs ease the problem of MBR rapid expanding and overlap by tightly bounding enclosed moving objects for some time in the future.

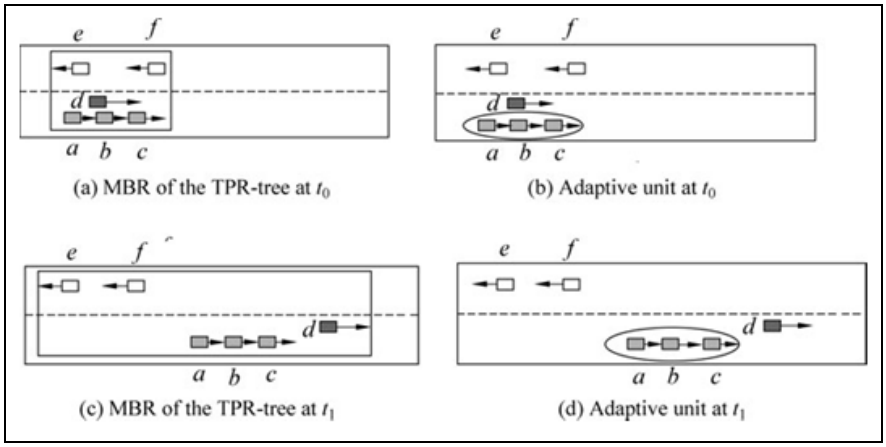

Figure 1: MBR vs. AU

We now formally introduce the AU. An AU is a 7-tuple:

AU* = (*auID,ob jSet,upperBound, lowerBound,
***edgeID,enterTime,exitTime*)**

Where *auID* is the identifier of the AU, *ob jSet* is a list that stores objects belonging to the AU, and *upperBound* and *lowerBound* are upper and lower bounds of predicted future trajectory of the AU. The trajectory bounds are derived from the trajectory bounds of the objects in the AU.

Assume the functions of trajectory bounds as follows:

***upperBound* : $Du(t) = \alpha u + \beta u \cdot t$**

***lowerBound* : $Dl(t) = \alpha l + \beta l \cdot t$**

edgeID denotes the edge that the AU belongs to, and *enterTime* and *exitTime* record the time when the AU enters and leaves the edge.
In the spatial network, multiple AUs are associated with a network edge. Since AUs in the same edge are likely to be accessed together during query processing, we store AUs by clustering on their *edgeID*. In other words, the AUs in the same edge are stored in the same disk pages. To access AUs more efficiently, create a compact summary structure called the ***direct access table*** for each edge, which is treated as a secondary index of AUs that can be accessed by an in-memory buffer. A direct access table stores the summary information of each AU on an edge (i.e., number of objects, trajectory bounds) and pointers to AU disk pages. Each AU corresponds to an entry in the direct access table, which has the following structure (*auID,upperBound, lowerBound,auPtr,ob jNum*), where *auPtr* points to a list of AUs in disk storage and *ob jNum* is the number of objects included in the AU.
Similar to AUs, the entries of the same direct access table and of the different direct access table but in the adjacent edge are grouped together so that we can get them into the buffer more efficiently. For a simple network with small amount of AUs, can retain all direct access tables in the main memory since it only keeps the summary information of AUs. When the updated locations are stored in a database in the server, the index structure of moving objects may be updated frequently with the update of locations. Our task is to reduce the cost of such indexing updates by a 1-dimensional dynamic AU structure and an accurate prediction method.
An important feature of the AU is that it groups objects having similar moving patterns. The AU is capable of dynamically adapting itself to cover the movement of the objects it contains. By tightly bounding enclosed moving objects for some time in the future, the AU reduces the update problem of MBR rapid expanding and overlaps in the TPR-tree like methods.

4.2. The Adaptive Network R-Tree (ANR-Tree)
Build a spatial index (e.g., R-tree) for the road network over the adaptive units to form the Adaptive Network R-tree (ANR-tree) for network-constrained moving objects. The ANR-tree is a two-level index structure. At the top level, it consists of a 2D R-tree that indexes the spatial information of the road network. At the bottom level, its leaves contain the edges representing multiple road segments (i.e., polylines) included in the corresponding MBR of the R-tree and these leaves point to the lists of adaptive units. Each entry in a leaf node consists of a road segment, i.e., a line segment in the polyline.
The top level R-tree remains fixed during the lifetime of the index scheme (unless there are changes in the network). The index scheme is developed with the R-tree in this study, but any existing spatial index can also be used without change Figure 2 shows the structure of the ANR-tree, which also includes a direct access table. The R-tree, the direct access table, and adaptive units are stored in the disk. However, the direct access table stores the summary information of some AUs on the edge and is similar to a secondary index of adaptive units. In the index scheme, each leaf node of the R-tree can be associated with its direct access table by its *edgeID* and the direct access table can connect to corresponding adaptive units by *auPtr* in its entries. Therefore, we only need to update the direct access table when AUs change, which enhances the performance of the index scheme.
Since the R-Tree indexes the road network, it remains fixed, and the update of the ANR-tree is restricted to the update of adaptive units. Specifically, an AU is usually created at the start of one edge and dropped at the end of the edge.

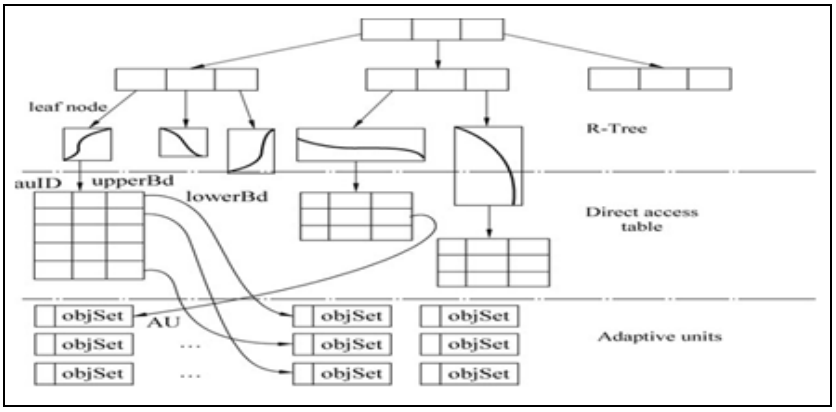

Figure 2: Structure of the ANR-tree

Since the AU is a 1-dimensional structure, it performs update operations much more efficiently than the 2-dimensional indexes. The update of the ANR-tree can be done as follows:
Creating an AU, dropping an AU, adding objects to an AU, and removing objects from an AU. Here the descriptions of these operations in detail.

4.2.1. Creating an AU
Actually, AU is created in two cases: (1) at the initial time when on bulk-loading at each network edge, and (2) when the objects leave the original edge with a single object. The steps to create an AU are:

- **Stage (i)** First create the ob jSet - a list of objects traveling in the same direction with similar velocities (velocity difference is not larger than a speed threshold), and in close-by locations (location difference is not larger than a distance threshold).

- **Stage (ii)** Then predict the future trajectories of the AU by simulation and calculate its trajectory bounds. In fact, we treat the AU as one moving object (the object closest to the center of the AU) and predict its future trajectory bounds by predicting this object.
- **Stage (iii)** The prediction starts when the AU is created and ends at the end of the edge.
- **Stage (iv)** Finally, we write the created AU to the disk page and insert the AU entry to its outline structure.

4.2.2. Adding and removing objects from an AU

- **Step (i)** When an object leaves an AU, remove this object from the AU and find another AU in the neighborhood to check if the object can fit that AU. If it can, the object will be inserted into that AU; otherwise, a new AU is created for this object.
- **Step (ii)** when adding an object into an AU, first find the direct access table of the edge in which the object lies and by its AU entry in the table, access the AU disk storage.
- **Step (iii)** Finally, insert the object into the objects list of the AU and update the AU entry in the direct access table.

Note: Removing an object from an AU involves a similar process.

The update* Algorithm *of the ANR-tree can be done as follows:

Update(ob jID, position, velocity,edgeID)
input : ob jID is the object identifier, position and velocity are its position and velocity,
edgeID is the edge identifier for the position of the object
Find AU where ob jID is included before update;
if AU.edgeID != edgeID or (ob jID is not the boundary object of AU and
(position < AU.lowerBound or position > AU.upperBound)) then
//The object moves to a new edge or exceeds bounds of its original AU;
Find the nearest AU AU1 for ob jID on edgeID;
if GetNum(AU1.ob jSet) < MAXOBJNUM and
ObjectFitAU(ob jID, position, velocity,AU1) then
InsertObject(ob jID,AU1.auID,AU1.edgeID);
else AU2 ← CreateAU(ob jID,edgeID);
if GetNum(AU.ob jSet) > 1 then
DeleteObject(ob jID,AU.auID,AU.edgeID);
else DropAU(AU.edgeID,AU.auID);
else if (ob jID is the low boundary object of AU and position−AU.lowerBound ≥ε) or
(ob jID is the high boundary object of AU and AU.upperBound− position ≥ε) then
AdaptAUBounds(AU, position);

5. Conclusion and Future Enhancement

In summary, updating the AU-based index is easier than updating the TPR-tree. It never invokes any complex node splitting and merging. Moreover, owing to the similar movement features of objects in an AU and the accurate prediction of the SP method, the objects are seldom removed or added from their AU on an edge, which reduces the number of index updates. Since the future movement of the adaptive unit is predicted through the simulation based prediction method, the ANR-tree can be used to index the future trajectory of moving objects to support efficient predictive queries. The predicted movement of the adaptive unit in the ANR-tree is not given by a single trajectory, but instead by two trajectory bounds based on different assumptions on the traffic conditions and obtained from the simulation. Since objects in an AU have similar movement, we then predict the movement of the AU, as if it were a single moving object. In this part, we propose future enhancement in an algorithm for predictive range query in the ANR-tree. It can also be extended to support the predictive *k*NN query and continuous query. A predictive range query effectively captures all objects moving in a road network. This will helpful to minimize the updating cost and increase the efficiency of querying future trajectories in a moving object.

6. References

1. Pfoser, D: Indexing the Trajectories of Moving Objects. IEEE Data Engineering Bulletin, 25(2):2-9, 2002
2. Agarwal PK, Arge L, Erickson J (2000) Indexing Moving Points. In: Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2000), Dallas, Texas, USA, pp 175-186
3. Almeida VT, G¨uting RH (2004) Indexing the Trajectories of Moving Objects in Networks. GeoInformatica 9(1):33-60
4. Indexing the Past, Present and Future Positions of Moving Objects on Fixed Networks Ying Fang, Jiaheng Cao, Yuwei Peng, Liwei Wang International Conference on Computer Science and Software Engineering (2008)
5. MOIST: A Scalable and Parallel Moving Object Indexer with School Tracking Google Research
6. Frentzos E (2003) Indexing Objects Moving on Fixed Networks. In: Proceedings of the 8th International Symposium on Spatial and Temporal Databases (SSTD 2003), Santorini Island, Greece, pp 289-305
7. Guttman A (1984) A Dynamic Index Structure for Spatial Searching. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 1984), Boston, Massachusetts, USA, pp 47-57

8. Jensen CS, Lin D, Ooi BC (2004) Query and Update Efficient B+Tree Based Indexing of Moving Objects. In: Proceedings of the 30th International Conference on Very Large Data Bases (VLDB 2004)
9. Kollios G, Gunopulos D, Tsotras VJ (1999) Effective Density Queries on Continuously Moving Objects. In: Proceedings of the 22nd International Conference on Data Engineering (ICDE 1999)
10. [Nascimento MA, Silva JRO (1998) Towards Historical R-trees. In: ACM Symposium on Applied Computing (SAC 1998), Atlanta, Georgia, USA, pp 235-240
11. Pfoser D, Jensen CS, Theodoridis Y (2000) Novel Approaches in Query Processing for Moving Object Trajectories. In: Proceedings of the 26th International Conference on Very Large Data Bases (VLDB 2000), Cairo, Egypt, pp 395-406
12. Pfoser D, Jensen CS (2003) Indexing of Network Constrained Moving Objects. In: Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems (GIS 2003), New Orleans, Louisiana, USA, pp 25-32
13. Saltenis S, Jensen CS (2002) Indexing of Moving Objects for Location-Based Service. In: Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, California, USA, pp 463-472
14. Y. Tao, D. Papadias, and J. Sun. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In VLDB, pages 790–801, 2003.
15. Tao Y, Papadias D (2001) The MV3R-Tree: A Spatiotemporal Access Method for Timestamp and Interval Queries. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001), Roma, Italy, pp 431-440
16. Tao Y, Papadias D, Sun J (2003) The TPR*-Tree: An Optimized Spatiotemporal Access Method for Predictive Queries. In: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB 2003), Berlin, Germany, pp 790-801.
17. Yiu ML, Tao Y, Mamoulis N (2006) The Bdual -Tree: Indexing Moving Objects by Space- Filling Curves in the Dual Space. The International Journal on Very Large Data Bases 17(3):379-400