

Indexing Future Trajectories of Moving Objects in a Constrained Network

Jidong Chen^{1,2}, Xiaofeng Meng^{1,2}

¹ School of Information, Renmin University of China, Beijing 100872, P. R. China

² Key Laboratory of Data Engineering and Knowledge Engineering, MOE, Beijing 100872, P. R. China

E-mail: {chenjd, xfmeng}@ruc.edu.cn

Abstract Advances in wireless sensor networks and positioning technologies enable new applications monitoring moving objects. Some of these applications, such as traffic management, require the possibility to query the future trajectories of the objects. In this paper, we propose an original data access method, the ANR-tree, which supports predictive queries. We focus on real life environments, where the objects move within constrained networks, such as vehicles on roads. We introduce a simulation-based prediction model based on graphs of cellular automata, which makes full use of the network constraints and the stochastic traffic behavior. Our technique differs strongly from the linear prediction model, which has low prediction accuracy and requires frequent updates when applied to real traffic with velocity changing frequently. The data structure extends the R-tree with adaptive units which group neighbor objects moving in the similar moving patterns. The predicted movement of the adaptive unit 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. Our experiments, carried on two different datasets, show that the ANR-tree is essentially one order of magnitude more efficient than the TPR-tree, and is much more scalable.

Keywords Database, Spatial Database, Access Methods, Moving Objects

1 Introduction

The continued advances in wireless sensors and positioning technologies have enabled a variety of new applications such as traffic management, fleet management, and location-based services that

monitor continuously changing positions of moving objects [12, 13]. Queries on moving objects can be divided into two categories: queries of historical locations of the moving objects, and queries of their anticipated future locations (a.k.a. predictive queries). In this paper, we are concerned with the second type of queries.

Many indexing techniques have been proposed to support predictive queries [1, 6, 7, 10, 14, 16], and most of them use a linear prediction model,

* Partly supported by the National Natural Science Foundation of China (Grant No. 60573091), the Key Project of Ministry of Education of China (Grant No. 03044), Program for New Century Excellent Talents in University (NCET), Program for Creative PhD Thesis in University

which relates objects positions as a linear function of time, to estimate their future positions. These predictive index structures are designed to index objects performing free movement in a two-dimensional space. They assume that the movement of the objects is unconstrained and is independent of each other as well as of the environment. However, in the real world, objects move within spatially constrained networks, e.g. vehicles move on road networks, and trains on railway networks. Furthermore, road situations may have an effect on the speed (e.g. in traffic jams) and the orientation (e.g. at junctions) of the vehicles. Overlooking this reality often leads to unrealistic data modeling and inaccurate prediction.

Unfortunately, current indexing work that handles network-constrained moving objects [2, 4, 12] is mostly concerned with historical movement. In addition, the linear models used in the predictive index structures cannot reflect the real movement. Applying linear prediction models on road traffic will lead to low prediction accuracy, which in turn, will lead to frequent updates.

In this paper, we introduce a new data model and access method to support predictive queries on moving objects in constrained networks. Our method makes full use of constraints of the network and the stochastic behavior of the real traffic, aiming to achieve high update/query efficiency. Our contribution is twofold. First, we propose a simulation-based prediction model which captures traffic features. Second, based on this model, we propose a new access method - the ANR-tree

that supports efficient predictive queries, and is robust for frequent updates. The ANR-tree extends the R-tree with adaptive units which group neighbor objects moving in the similar moving patterns. The predicted movement of the adaptive unit 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. We have carried on complexity analysis of the algorithms in terms of I/O and carried out extensive experiments based on two datasets. The results show the efficiency of the ANR-tree.

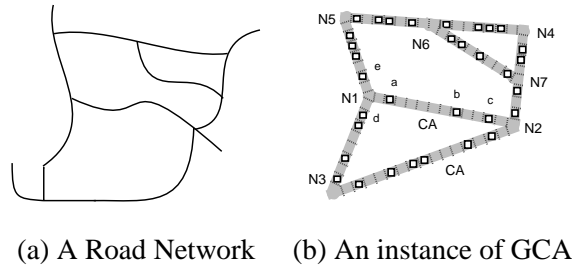
The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 presents the simulation-based prediction model. In Section 4, the ANR-tree is described with its data structure and algorithms. Section 5 contains algorithm analysis and experimental evaluation. We conclude in Section 6.

2 Related Work

Research on spatio-temporal access methods has focused on two issues: (i) storage and retrieval of historical information, (ii) prediction of future trajectory. 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 [9, 11, 15], all based on 3-dimensional variations of R-trees [5], have been proposed, and their goal is to minimize storage and query cost. To manage moving

objects in spatially constrained networks, Pfoser et al [12] proposed to convert the 3-dimension problem into two sub-problems of lower dimensions through certain transformation of the networks and the trajectories. Another approach, known as the FNR-tree [4], separates spatial and temporal components of the trajectories and indexes the time intervals that any moving object was on a given network link. The MON-tree approach [2] 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.

Indexing future trajectories raises different problems than 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. Some of the early works [1, 7] employ dual transformation techniques that represent predicted positions as points moving in a 2-dimensional space. However, they are largely theoretical, and applicable either only in 1D space [7] or entirely inapplicable in practice due to some large hidden constant in complexity [1]. Also based on dual transformation, a recent approach called STRIPES [10] supports efficient query and update at the cost of increased space requirements. The main focus of most recent work is on practical implementation, for instance, the TPR-tree [14] and its variations [13, 16] are based on R-trees [5], and the B^x -tree [6] and its variations [18] based on B^+ -tree. These structures use the linear prediction model to support the predictive query and to reduce the number of index up-



(a) A Road Network (b) An instance of GCA

Figure 1: An example of a road network and its GCA model

dates. However, the assumption of linear movement limits their applicability in a majority of real applications especially in traffic network where vehicles change their velocities frequently. To the best of our knowledge, no current indexing method supports predictive queries of network constrained moving objects.

3 Data Model and Trajectory Prediction

We model a road network with a graph of cellular automata (GCA), where the nodes of the graph represent road intersections and the edges represent road segments with no intersections. Each edge consists of a cellular automaton (CA), which is represented, in a discrete mode, as a finite sequence of cells. Figure 1 shows an example of a road network and its GCA model. Each node has a label which represents an intersection of the road network. The wide lines represent edges and each edge treated as one CA connects many cells.

In the GCA, a moving object is represented as a symbol attached to the cell and it can move several cells ahead at each time unit. Intuitively, the velocity is the number of cells an object can traverse

during a time unit.

Let i be an object moving along an edge. Let $v(i)$ be its velocity, $x(i)$ its position, $gap(i)$ the number of empty cells ahead (forward gap), and $P_d(i)$ a randomized slowdown rate which specifies the probability it slows down. We assume that V_{max} is the maximum velocity of moving objects. The position and velocity of each object might change at each transition of the GCA according to the rules below (adapted from [8]):

1. if $v(i) < V_{max}$ and $v(i) < gap(i)$ then $v(i) \leftarrow v(i) + 1$
2. if $v(i) > gap(i)$ then $v(i) \leftarrow gap(i)$
3. if $v(i) > 0$ and $rand() < P_d(i)$ then $v(i) \leftarrow v(i) - 1$
4. if $(x(i) + v(i)) \leq L$ then $x(i) \leftarrow x(i) + v(i)$

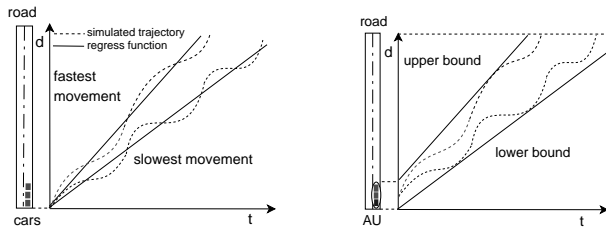
The first rule represents linear acceleration until the object reaches the maximum speed V_{max} . The second rule ensures that if there is another object in front of the current object, it will slow down in order to avoid collision. In the third rule, $P_d(i)$ models erratic movement behavior. Finally, the new position of object i is given by the fourth rule as sum of the previous position with the new velocity if it is in the CA.

We use GCAs not only to model road networks, but also to simulate the movements of moving objects by the transitions of the GCA. Based on the GCA, a *Simulation-based Prediction (SP)* model to anticipate future trajectories of moving objects

is proposed. The SP model treats the objects simulated results as their predicted positions. To refine the accuracy, based on different assumptions on the traffic conditions we simulate two future trajectories in discrete points for each object. Then, by linear regression and translating, the trajectory bounds that contain all possible future positions of a moving object can be obtained. The process of the simulation-based prediction can be seen in Figure 2.

Most existing work uses the CA model for traffic flow simulation in which the parameter $P_d(i)$ is treated as a random variable to reflect the stochastic, dynamic nature of traffic system. However, we extend this model for predicting the future trajectories of objects by setting $P_d(i)$ to values that model different traffic conditions. For example, laminar traffic can be simulated with $P_d(i)$ set to 0 or a small value, and the congestion can be simulated with a larger $P_d(i)$. By giving $P_d(i)$ two values, we can derive two future trajectories, which describe, respectively, the fastest and slowest movements of objects as showed in Figure 2(a). In other words, the object future locations are most probably bounded by these two trajectories. The value of $P_d(i)$ can be obtained by the experiences or by sampling from the given dataset.

Through the SP model, we obtain two bounds of objects future trajectory. In the sequel, we apply this technique in our index to a set of moving objects that have similar movement and are treated as one object.



(a) Simulated trajectories (b) Future trajectory bounds

Figure 2: The simulation-based prediction

4 The ANR-tree

4.1 Structure and Storage

The Adaptive Network R-tree (ANR-tree) is a two-level index structure. At the top level, it consists of a 2D R-tree that indexes spatial information of the road network. On the bottom level, its leaves contain the cellular automata representing road segments included in the corresponding MBR of the R-tree and point to the lists of adaptive units, with objects future trajectories. The top level R-Tree remains fixed during the lifetime of the ANR-Tree (unless there are changes in the network). The paper is developed with R-trees, but any existing spatial index can also be used without changes.

An important feature of the ANR-tree is that it groups objects having similar moving patterns into adaptive units (AUs). Conceptually, an adaptive unit is similar to a one-dimensional MBR in the TPR-tree, that expands with time according to the predicted movement of the objects it contains. 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 overlaps with other MBRs. The AU avoids this problem by grouping adjacent objects with the same direction and similar speed according to a distance threshold and a speed threshold. The AU is capable of dynamically adapting itself to cover the movement of the objects it contains. Since objects in an AU have similar movement, we then predict the movement of the AU, as if it were a single moving object.

We now formally introduce the AU. An AU is a 8-tuple:

$$AU = (\text{auID}, \text{objSet}, \text{upperBound}, \text{lowerBound}, \text{edgeID}, \text{enterTime}, \text{exitTime}, \text{auInitLen})$$

where `auID` is the identifier of the AU, `objSet` is a list that stores objects belonging to the AU, `upperBound` and `lowerBound` are upper and lower bounds of future trajectory of the AU. The trajectory bounds of the AU are derived from the trajectory bounds of the objects in the AU. We assume the functions of the trajectory bounds as follows:

$$\text{upperBound} : D(t) = \alpha_u + \beta_u \cdot t$$

$$\text{lowerBound} : D(t) = \alpha_l + \beta_l \cdot t$$

`edgeID` denotes the edge the AU belongs to, `enterTime` and `exitTime` record the time when the AU enters and leaves the edge and `auInitLen` represents the initial length.

In the road 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 cluster-

ing on their `edgeID`. That is, the AUs in the same edge are stored in the same disk pages. To access AUs more efficiently, we create an in-memory, compact summary structure called the *direct access table* for each edge. 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`, `objNum`), where `auPtr` points to a list of AUs in disk storage and `objNum` is the number of objects. In order to minimize I/O cost, we use the direct access table to filter AUs and only access the disk pages when necessary.

Figure 3 shows the structure of the ANR-tree. The R-tree and adaptive units in the ANR-tree are stored in the disk. However, the direct access table is in the main memory (also partly in the disk for a large road network) since it only keeps the summary information of AUs. In the ANR-tree, 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 greatly enhances the index performance.

4.2 Query Algorithm

In this part, we propose an algorithm for predictive range query in the ANR-tree. It can also be ex-

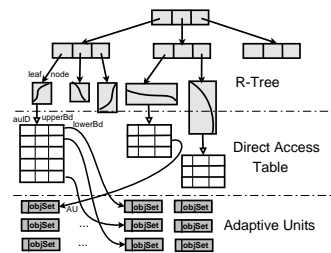
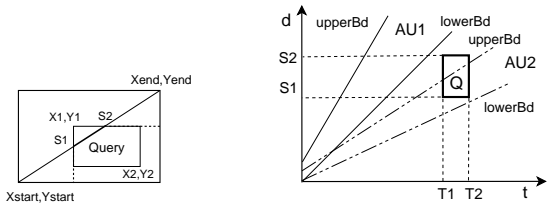


Figure 3: Structure of the ANR-tree

tended to support the (K) Nearest Neighbor query and continuous query. A predictive range query captures all objects whose locations are inside a specified region R during time interval $[T_1, T_2]$ in the future. Given a spatiotemporal window range with $(X_1, Y_1, X_2, Y_2, T_1, T_2)$, the query algorithm on the ANR-tree consists of the following steps:

- 1) We first perform a spatial window range search (X_1, Y_1, X_2, Y_2) in the top level R-Tree to locate the edges (e.g. e_1, e_2, e_3, \dots) that intersect the spatial query range.

- 2) For each selected edge e_i , we transform the original 3D search $(X_1, Y_1, X_2, Y_2, T_1, T_2)$ to a 2D search (S_1, S_2, T_1, T_2) ($S_1 \leq S_2, T_1 \leq T_2$), where S_1 and S_2 are the relative distances from the start vertex along the edge e_i . Figure 4(a) gives an example when the query window range only intersects one edge. In the case of multiple intersecting edges, we can divide the query range into several sub-ranges by edges and apply the transformation method to each edge. The method is also applicable to the various modes the query and edges intersect. For space limitation, we only illustrate the case in Figure 4(a) and compute its relative distances S_1 and S_2 . It can be easily extended to



(a) Query transformation (b) Query process in AUs

Figure 4: Window range query in the ANR-tree

other cases. Suppose $X_{start}, Y_{start}, X_{end}, Y_{end}$ are the start vertex coordinates and the end vertex coordinates of the edge e_i . According to Thales Theorem about similar triangles, we obtain S_1 and S_2 as follows:

$$r = \sqrt{(X_{start} - X_{end})^2 + (Y_{start} - Y_{end})^2}$$

$$S_1 = \frac{X_1 - X_{start}}{X_{end} - X_{start}} r$$

$$S_2 = \frac{Y_1 - Y_{start}}{Y_{end} - Y_{start}} r$$

3) We further find the adjacent edges on which objects are possible to move into the window range during the future period $[T_1, T_2]$. Specifically, it expands the network from the edge points intersecting the spatial window (e.g. locations of S_1, S_2) up to a maximum distance computed by the future time and the average maximum speed in the network and returns the traversed edges (e.g. e'_1, e'_2, e'_3, \dots). In this way, we can avoid the false misses during the query processing.

4) The transformed query (S_1, S_2, T_1, T_2) is executed in each of the AUs in the direct access table of the corresponding edge e_i or adjacent edge e'_i . As illustrated by Figure 4(b), an AU is suitable to the query only if the 2D window range intersects the area between the upper and lower trajectory bounds of the AU. Otherwise when the query

is below the lower bound (e.g. $\beta_l * T_1 + \alpha_l > S_2$) or above the upper bound (e.g. $\beta_u * T_2 + \alpha_u < S_1$) of the AU, the query cannot contain objects in this AU. In our example, the query only returns AU2. By the trajectory bounds of the AU, we can determine whether the transformed query intersects the AU, thus filtering out the unnecessary AUs quickly.

5) Finally, we access the selected AUs in disk storage and return the objects satisfying the predictive query window.

4.3 Update Operations

Since the top level R-Tree indexes the road network, it remains fixed, and the update of the ANR-tree restricts to the update of AUs. The update of an AU contains creating an AU, dropping an AU, adding objects to an AU and removing objects from an AU. Specifically, an AU is usually created at the start of one edge and dropped at the end of the edge. To create an AU, we first compose the *objSet*, a list of objects traveling in the same direction with similar velocities and in close-by locations (computed by a distance threshold and a velocity threshold). We then predict the future trajectories of the AU by simulation and compute its trajectory bounds. Since the AU is a one-dimensional structure, the ANR-tree performs update operations much more efficiently than two-dimensional indexes.

When updating an object in the ANR-tree, we first determine whether the object is leaving the edge and entering another one. If it is moving to another edge, we delete it from the old AU (if it

is the last object in the old AU, the AU is also dropped) and insert it into the nearest AU or create a new AU in the edge it is entering. Otherwise, we do not update the AU that the object belongs to unless its position exceeds the bounds of the AU. In that case, we execute the same updates as those when it moves to another edge. Factually, we find, from the experiment evaluation, that the chances that objects move beyond the trajectory bounds of its AU are very slim.

5 Performance Analysis

In this section, we first analyze the I/O cost of the query algorithms and then perform experimental evaluation.

5.1 Algorithms Analysis

We follow the main assumptions of [17] in our analysis, in particular we assume that rectangles, including the whole map, are square. Let M be the total number of edges of the GCA, W be the width of the map, N be the total number of objects and n be the average number of objects in an AU. The average number of AUs in an edge is N/Mn . We assume that B is the maximum number of objects in a disk page. The average number of AUs in a page is B/n .

For a spatiotemporal query window $(X_1, Y_1, X_2, Y_2, T_1, T_2)$, a spatial search is first performed in the top level R-tree to locate the edges that intersect the spatial window. Let N_r be the number of data rectangles of the R-tree, f be its average fanout, $h = 1 + \lceil \log_f \frac{N_r}{f} \rceil$ its

height, and $S_{l,x}, S_{l,y}$ the average extents of node rectangles at level l on X and Y coordinates. Assume that each node is in one disk page, the average number of disk accesses for the spatial search (X_1, Y_1, X_2, Y_2) is given by [17]:

$$\sum_{l=1}^{h-1} \frac{N_r}{f^l} \frac{(S_{l,x} + |X_2 - X_1|)(S_{l,y} + |Y_2 - Y_1|)}{W^2}$$

Since each leaf node of the R-tree only contains one edge, the average number of edges intersecting the spatial query is given by:

$$M \frac{(S_{1,x} + |X_2 - X_1|)(S_{1,y} + |Y_2 - Y_1|)}{W^2}$$

For each selected edge, we scan its direct access table for the purpose of only accessing relevant AUs. We compute the average number of AUs intersecting the transformed query (S_1, S_2, T_1, T_2) . In Figure 2(b), the two trajectory bounds of one AU divide the coordinate plane into three parts: upper area (above the upper bound), middle area (between the upper and lower bounds) and lower area (below the lower bound). We assume that the upper and lower areas represent respectively the percentage of μ_u, μ_l of the total area of the plane. The probability that the query intersects the AU is $(1 - \mu_u^2 - \mu_l^2)$. It is not difficult to compute the average probabilities $\bar{\mu}_u, \bar{\mu}_l$ of the AUs on the edge using their bound functions and the length of the edge. Now, we can get the average number of relevant AUs as follows:

$$(1 - \bar{\mu}_u^2 - \bar{\mu}_l^2) \frac{N}{Mn}$$

Finally, for each relevant AU, we need to find the moving objects satisfying the predictive query range. Since the AUs on the same edge are likely clustered in the same disk page, the average I/O cost of accessing relevant AUs and moving objects on each selected edge is given by:

$$(1 - \bar{\mu}_u^2 - \bar{\mu}_l^2) \frac{N}{MB}$$

Therefore, the total I/O cost for a spatiotemporal window query in the ANR-tree is given by:

$$\frac{1}{W^2} \left(\sum_{l=1}^{h-1} \frac{N_r}{f^l} (S_{l,x} + |X_2 - X_1|)(S_{l,y} + |Y_2 - Y_1|) \right) + \frac{N}{B} (S_{1,x} + |X_2 - X_1|)(S_{1,y} + |Y_2 - Y_1|)(1 - \bar{\mu}_u^2 - \bar{\mu}_l^2)$$

5.2 Experimental Evaluation

We evaluate the ANR-tree by comparing it with the TPR-tree and the ANR-tree when the direct access table is not used (denoted as ‘‘ANR-tree without DT’’). We measure their query performance with different data set size, updates and query window size and their update performance.

5.2.1 Datasets

We use two datasets for our experiments. The first is generated by the CA simulator, and the second by the Brinkhoff Network-based Generator [3]. We use the CA traffic simulator to generate a given number of objects in a uniform network of size 10000×10000 consisting of 500 edges. Each object has its route and is initially placed at a random position on its route. The initial velocities of the objects follow a uniform random distribution in the range $[0, 30]$. The location and

velocity of every object is updated at each timestamp. The Brinkhoff Network-based Generator is used as a popular benchmark in many related work. The generator takes a map of a real road network as input (our experiment is based on the map of Oldenburg including 7035 edges). The positions of the objects are given in two dimensional X-Y coordinates. We transform them to the form of (edgeid, pos), where edgeid denotes the edge identifier and pos denotes the object relative position on the edge. The generator places a given number of objects at random positions on the road network, and updates their locations at each timestamp. We implemented both the ANR-tree and the TPR-tree in Java and carried out experiments on a Pentium 4, 2.4 GHz PC with 512MB RAM running Windows XP. To improve the performance of the ANR-tree, we employed a LRU buffer of the same size as the one used in the TPR-tree [14].

5.2.2 Query Cost

Effect of Data Size We study the window range query performance of the TPR-tree and the ANR-tree while varying the number of moving objects from 10k to 100k. Figure 5 shows the average number of I/O per query. In each case, the query cost increases as the data size increases. However, the ANR-tree has much lower cost than the TPR-tree. This is because the adaptive units in the ANR-tree have much less overlaps than the MBRs in the TPR-tree, and the overlaps to a large extent determine the range query cost. The ANR-tree with the direct access table achieves better performance

than the ANR-tree without it. This is because the in-memory summary structure enables us to filter some unnecessary AUs during the search of AUs that intersect the range query. However, for the Brinkhoff dataset the benefit of the direct access table is not obvious because the large number of small edges in the network reduces chances of filtering the AUs not included in the range query.

For each data size, the search costs of the two indices in the Brinkhoff dataset are both higher than those in the CA dataset due to the higher complexity of road network and skewed spatial distribution of objects in the Brinkhoff dataset. This happens in each experiment.

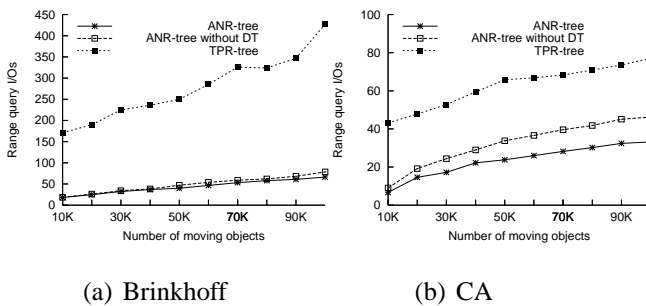


Figure 5: Effect of Data Size on Query

Effect of Query Window Size To study the effect of query window size on performance, we increase the window size from 10 to 100 for 100K data size with a workload of 200 range queries. Figure 6 shows the query cost as a function of the query window size. It is clear that for all the index solutions, query cost increases with the query window size. This is so because larger windows contain more objects and therefore lead to more node accesses. However, this effect is more obvious on the TPR-tree.

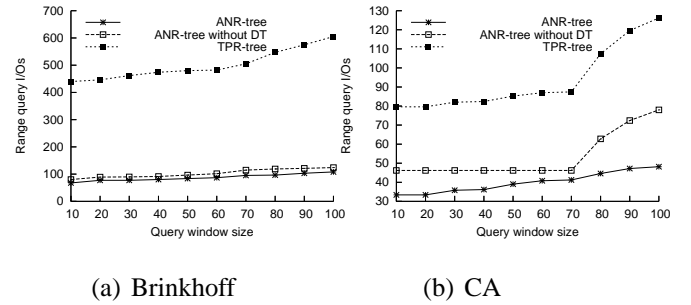


Figure 6: Effect of Query Window Size on Query

Effect of Update We increase the number of updates from 100K to 1M to examine how query performance is affected. We issued 200 range queries for every 100K updates in a 1M dataset. Figure 7 shows that the cost of the TPR-tree increases much faster as the number of updates increases. The cost of the ANR-tree is considerably lower and is less sensitive to the number of updates. This is because as objects move apart, the amount of dead space in the TPR-tree increases, which makes false hits more likely. Also, updates lead to the expanding and overlaps of MBRs, which further deteriorate the performance of the TPR-tree. For the ANR-tree, the increase of the updates hardly affect the total number of AUs, and the chances of overlaps of different AUs are very slim.

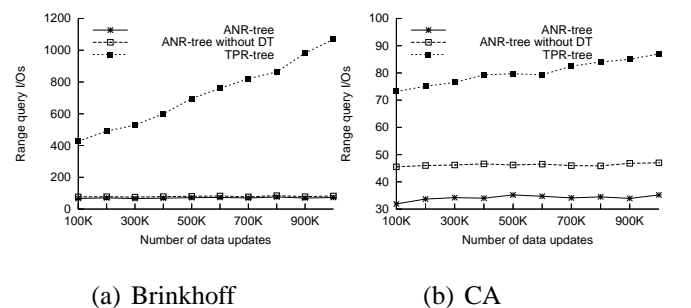


Figure 7: Effect of Updates on Query

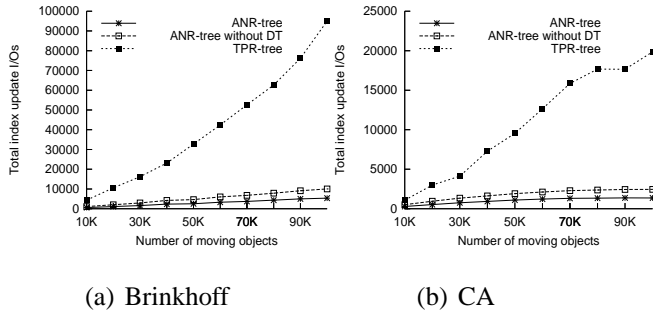


Figure 8: Update Cost with Datasize

5.2.3 Update Cost

We study the update performance of the index with data size. The update costs depend on the update frequency and the average individual update cost. From Figure 8, we can see that although the ANR-tree has to deal with the creation and dropping of AUs, the TPR-tree incurs much higher update costs than the ANR-tree and its performance deteriorates dramatically as data size increases. This is mainly due to the inaccuracy of the linear prediction model and the complex reconstruction of the TPR-tree (e.g. splitting and merging).

6 Conclusion

Querying future trajectories of objects moving in a constrained network is a topic of great practical importance. Our contribution is twofold. First we propose a prediction model, based on simulation, which predicts with a great accuracy the future trajectories of moving objects. The accuracy results from the fact that the model exploits the constraints of the network and models the stochastic aspect of urban traffic. Then, we propose a new access structure, which extends the R-tree with adaptive units that exploit as much as possible the move-

ment characteristics of objects. The efficiency of the structure results from the possible reduction of 3D spatiotemporal queries to 2D queries. Our experimental results performed on two datasets show that the efficiency of our algorithm is one order of magnitude higher than the TPR-tree.

Acknowledgments

The authors would like to thank Haixun Wang from IBM T. J. Watson Research, Karine Zeitouni from PRISM, Versailles Saint-Quentin University in France and Stéphane Grumbach from CNRS, LIAMA China for many helpful advice.

References

- [1] Agarwal P K, Arge L, Erickson J. Indexing moving points (extended abstract). In *Proc. of the 19th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of database systems*, Dallas, Texas, 2000, pp.175-186.
- [2] Almeida V T D, Güting R H. Indexing the Trajectories of Moving Objects in Networks. *GeoInformatica*, 2005, 9(1):33-60.
- [3] Brinkhoff T. A framework for generating network-based moving objects. *GeoInformatica*, 2002, 6(2):153-180.
- [4] Frentzos E. Indexing objects moving on fixed networks. In *Proc. of the 8th Intl. Symp. on Spatial and Temporal Databases*, Santorini Island, Greece, 2003, pp.289-305.
- [5] Guttman A. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, Boston, Massachusetts, 1984, pp.47-57.
- [6] Jensen C S, Lin D, Ooi B C. Query and Update Efficient B+-Tree Based Indexing of Moving Objects. In *Proc. of 30th Int. Conf. on Very Large Data Bases*, Toronto, Canada, 2004, pp.768-779.
- [7] Kollios G, Gunopulos D, Tsotras J V. On indexing mobile objects. In *Proc. of the 8th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of database systems*, Philadelphia, Pennsylvania, 1999, pp.261-272.

- [8] Nagel K, Schreckenberg M. A cellular automaton model for freeway traffic. In *Journal Physique*, 1992, 2:2221-2229.
- [9] Nascimento M A, Silva J R O. Towards Historical R-trees. In *ACM Symposium on Applied Computing*, Atlanta, Georgia, 1998, pp.235-240.
- [10] Patel M, Chen Y, Chakka V. STRIPES: An Efficient Index for Predicted Trajectories. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, Paris, France, 2004, pp.637-646.
- [11] Pfoser D, Jensen C S, Theodoridis Y. Novel Approaches in Query Processing for Moving Object Trajectories. In *Proc. of 26th Int. Conf. on Very Large Data Bases*, Cairo, Egypt, 2000, pp.395-406.
- [12] 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.
- [13] 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.
- [14] Saltenis S, Jensen C S, Leutenegger S T, Lopez M A. Indexing the Positions of Continuously Moving Objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, Dallas, Texas, USA, 2000, pp.331-342.
- [15] Tao Y, Papadias D. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *Proc. of 27th Int. Conf. on Very Large Data Bases*, Roma, Italy, 2001, pp.431-440.
- [16] Tao Y, Papadias D, Sun J. The TPR*-Tree: An Optimized spatiotemporal Access Method for Predictive Queries. In *Proc. of 29th Int. Conf. on Very Large Data Bases*, Berlin, Germany, 2003, pp.790-801.
- [17] Theodoridis Y, Stefanakis E, Sellis T K. Efficient Cost Models for Spatial Queries Using R-Trees. *TKDE*, 2000, 12(1): 19-32.
- [18] Yiu M L, Tao Y, Mamoulis N. The B^{dual} -Tree: Indexing Moving Objects by Space-Filling Curves in the Dual Space. To appear in *Very Large Data Base Journal*, 2006.