Geoinformatica (2009) 13:397–424
DOI 10.1007/s10707-008-0052-5

Update-efficient indexing of moving objects in road networks

Jidong Chen · Xiaofeng Meng

Received: 22 December 2006 / Revised: 1 April 2008 /
Accepted: 30 April 2008 / Published online: 22 June 2008

Abstract Recent advances in wireless sensor networks and positioning technologies have boosted new applications that manage moving objects. In such applications, a dynamic index is often built to expedite evaluation of spatial queries. However, the development of efficient indexes is a challenge due to frequent object movement. In this paper, we propose a new update-efficient index method for moving objects in road networks. We introduce a dynamic data structure, called *adaptive unit*, to group neighboring objects with similar movement patterns. To reduce updates, an adaptive unit captures the movement bounds of the objects based on a prediction method, which considers road-network constraints and the stochastic traffic behavior. A spatial index (e.g., R-tree) for the road network is then built over the adaptive unit structures. Simulation experiments, carried on two different datasets, show that an adaptive-unit based index is efficient for both updating and querying performances.

Keywords Spatial-temporal databases · Moving objects · Index structure · Road networks

1 Introduction

Recent advances in wireless sensor networks and positioning technologies have enabled a variety of new applications such as traffic management, fleet management,

The corresponding author currently works at EMC Research China, Beijing, China.

J. Chen (✉) · X. Meng
School of Information, Renmin University of China, Beijing, China
e-mail: chen_jidong@emc.com

X. Meng
Key Laboratory of Data Engineering and Knowledge Engineering, MOE, Beijing, China
e-mail: xfmeng@ruc.edu.cn

and location-based services that manage continuously changing positions of moving objects [25], [27]. In such applications, a dynamic index is often built to expedite evaluation of spatial queries. However, existing dynamic index structures (e.g. B-tree and R-tree) suffer from poor performance due to the large overhead of keeping the index updated with the frequently changing position data. The development of efficient indexes to improve the update performance is an important challenge.

Current work on reducing the index updates of moving objects mainly contains three kinds of approaches. First, most efforts [9], [18], [19], [36] focus on the update optimization of the existing multi-dimensional index structures especially the adaptation and extension of the R-tree [12]. To avoid the multiple paths search operation in the R-tree during the top-down update, some recent works propose the bottom-up approach [18], [19] and memo-based [36] structure to reduce the updates of the R-tree. Another method [9] exploits the change-tolerant property of the index structure to reduce the number of updates that cross the minimized boundary rectangle (MBR) boundaries of the R-tree.

However, the indexes based on MBRs exhibit high concurrency overheads during node splitting, and each individual update is still costly. Therefore, some index methods based on a low-dimensional index structure (e.g. B^+-tree) are proposed [14], [37], which construct the second category of index methods. They combine the dimension reduction and linearization technique with a single B^+-tree to efficiently update the index structure.

The third kind of approaches use a prediction method with a time-parameterized function to reduce the index updates [27], [29], [32]. They describe a moving object's location by a linear function and the index is updated only when the parameters of the function change, for example, when the moving object changes its speed or direction. The MBRs of the index vary with the time as a function of the enclosed objects. However, it is hard for the linear prediction to reflect the movement in many real applications and therefore leads to a low prediction accuracy and frequent updates.

Though these index structures solve the problem of index updates to some extent, they are designed to index objects performing free movement in a two-dimensional space. We focus on the index update problem in real life environments, where the objects move within constrained networks, such as vehicles on roads. In such a setting, the spatial property of objects' movement is captured by the network and the static information can be separated from the dynamic information. Therefore, the spatial location of moving objects can be indexed by means of the road-network index structure. For example, moving objects can be accessed by each road segment which is indexed by the R-tree. Since the road network seldom changes and objects just move from one part to the other part of the network, the R-tree in this case remains fixed. Existing index work that handles network-constrained moving objects [3], [11], [25] is based on this feature. These works separate spatial and temporal components of the moving objects' trajectories and index the spatial aspect by the network with an R-tree. However, they are mostly concerned with the historical movements and therefore they do not consider the problem of index updates.

In this paper, we address the problem of efficient indexing of moving objects in road networks to support heavy loads of updates. We exploit the constraints of the network and the stochastic behavior of the real traffic to achieve both high updating and querying efficiency. We introduce a dynamic data structure, called *adaptive unit* (AU for short) 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 AUs to form the index scheme for moving objects in road networks. The index scheme optimizes the update performance for the following reasons. (1) An AU functions as a one-dimensional MBR in the TPR-tree [29], while it minimizes expanding and overlaps by considering more movement features. (2) The AU captures the movement bounds of the objects based on a prediction method, which considers the road-network constraints and stochastic traffic behavior. (3) Since the movement of objects is reduced to occur in one spatial dimension and attached to the network, the update of the index scheme is only restricted to the update of the AUs. Since the AU is approximated by its center object for efficiency, the query will possibly has false negative result. For improving it, we refine the prediction accuracy by simulating two trajectories based on different assumptions on the traffic conditions and revising the trajectory bounds when prediction accuracy decreases over time. We have carried out extensive experiments based on two datasets. The results show that an adaptive-unit based index not only improves the efficiency of each individual update but also reduces the number of index updates and is efficient for both updating and querying performance.

The main contributions of this paper are:

- The introduction of the graph of cellular automata (GCA) model and the simulation-based prediction (SP) model which capture traffic features and reduce the index updates.
- The introduction of AUs that optimize for frequent index updates and support the predictive query on moving objects in road networks.
- An experimental evaluation and validation of the efficient update as well as query performance of the proposed index structure.

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 introduces our data model and trajectory prediction method. Section 4 describes the structure and algorithms of AUs for efficient updates and query processing. Section 5 contains algorithm analysis and experimental evaluation. We conclude and propose the future work in Section 6.

2 Related work

Many efforts have been made on reducing the need for index updates of moving objects. In summary, they can be classified into three categories.

First, most work focuses on updating optimization of existing multi-dimensional index structures especially in the adaptation and extension of the R-tree [12]. The top-down update of an R-tree is costly since it needs several paths for searching the right data item considering the MBR overlaps. In order to reduce the overhead, Kwon et al. [18] develop the Lazy Update R-tree, which is updated only when an object moves out of the corresponding MBR. With adding a secondary index on the R-tree, it can perform the update operation in a bottom-up way. Recently, by exploiting the change-tolerant property of the index structure, Cheng et al. [9] present the CTR-tree to maximize the opportunity for applying lazy updates and reduce the number of updates that cross MBR boundaries. Lee et al. [19] extends the main idea of [18] and generalizes the bottom-up update approach. However, they are not suitable to the case where consecutive changes of objects are large. Xiong and Aref [36] present the RUM-tree that processes R-tree updates in a memo-based approach, which eliminates the need to delete the old data item during an index update. Therefore, its update performance is stable with respect to the changes between consecutive updates. In our index structure, however, the R-tree remains fixed since it indexes the road network and only the AUs are updated.

The second type of methods are based on the dimension reduction technique [25] and a low-dimensional index [14], [37] (e.g. B^+-tree). The B^x-tree [14], [37] combines the linearization technique with a single B^+-tree to efficiently update the index structure. It uses space filling curves and a pre-defined time interval to partition the representation of the locations of moving objects. This makes the B^+-tree capable of indexing the two-dimensional spatial locations of moving objects. Therefore, the cost of individual updating of index is reduced. However, the two-dimensional locations of objects are linearized by a space-filling curve and the time is also partitioned by a pre-defined time interval. Therefore, the B^x-tree imposes discrete representation and may not keep the precise values of location and time during the partitioning. For our setting, the two-dimensional spatial locations of moving objects can be reduced to the 1.5 dimensions [16] by the road network where objects move.

The techniques in the third category use a prediction method represented as the time-parameterized function to reduce the index updates [27], [29], [32]. They store the parameters of the function, e.g. the velocity and the starting position of an object, instead of the real positions. In this way, they update the index structure only when the parameters change (for example, the speed or the direction of a moving object changes). The Time-Parameterized R-tree (TPR-tree) [29] and its variants (e.g. TPR*-tree) [27], [32] are examples of this type of index structures. They all use a linear prediction model, which relates objects' positions as a linear function of the time. Actually, these methods also can support predictive queries that are usually processed by the dual transformation technique in some index methods [2], [23]. However, the linear prediction is hard to reflect the movement in many real application especially in traffic networks where vehicles change their velocities frequently. The frequent changes of the object's velocity will incur repeated updates of the index structure. Moreover, other prediction models with non-linear prediction proposed by Aggarwal et al. [1] using quadratic predictive functions and by Tao et al. [34] based on recursive motion functions for objects with unknown motion patterns improve the precision in predicting the location of each object, but they ignore the correlation of adjacent objects and may not reflect accurately some complex and stochastic traffic movement scenarios. Our techniques also fall into this category and apply an accurate prediction method when updating index structure by considering more transportation features. Our prediction method also can be applied to update policy of objects [8], a different research issue, which focuses on how to minimize the number of location updates sampled by sensors or GPS periodically from moving objects to the server database. However, for comparisons with the TPR-tree in our experiments, we use the same update policy and location representation of objects, but different prediction method in the index structure.

Compared to the TPR-tree, our technique will reduce the indexing update costs when the same amount of updated locations of one object are stored in a database in the server.

There is some work on data models, indexing and query processing for moving objects in road networks, which is also related to our work. Data models for network constrained moving objects have been a focus of recent study [10], [28], [35] because they form a foundation for data storage and query processing. Indexing techniques for objects moving in road networks also become a focus of study [3], [11], [17], [25]. Pfoser et al. [25] propose to convert a 3-dimensional problem into two sub-problems of lower dimensions through certain transformation of the networks and trajectories. Another approach, known as the fixed network R (FNR)-tree [11], separates spatial and temporal components of the trajectories and indexes time intervals that each moving object spends on a given network link. The MON-tree approach [3] 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. However, they all focus on the historical movement and cannot support frequent index updates. There are also other work [22], [24], [31], all based on 3-dimensional variations of R-trees [12] and R*-trees [4], to index the historical trajectory in Euclidean spaces. Their goal is to minimize storage and query cost, which does not consider the index update problem. Similar to our index structure, the IMORS method [17] focuses on reducing the number of index updates on a road network with the same idea of separating dynamic and static parts of an index structure. However, moving objects are indexed by a static small road sector blocks and may move to different sectors very soon, therefore their coordinates and bi-directional pointers to the road sector are likely to be updated frequently when their locations especially velocity have been changed. While the update performance can be improved by enlarging the length of a road sector, it may result in a degradation of the query processing performance. Instead, considering more traffic features, our AU index, a dynamic structure, groups objects having similar moving patterns and can dynamically adapt itself to cover the movement of the objects it contains by a more accurate prediction method. Therefore, it reduces more index updates both by road-network features and by a new prediction method. In addition, the AU index can also support the efficient predictive queries on road networks, which is not implemented in the IMORS method.

Query processing algorithms in spatial network databases have been developed using network distance [6], [13], [15], [26], [30]. Most of them only work for static data objects and do not monitor queries over moving objects in road networks. Mouratidis et al. [20] study the continuous monitoring of nearest neighbors in highly dynamic scenarios, where queries and data objects move frequently in the network. Similar to our work, they target frequent data updates to support the NN queries on moving objects in road networks. However, they store the network, objects and queries in three memory-resident data structures: a spatial index on the network edges, an edge table maintaining network and moving objects, and a query table with expansion tree for each query. To incrementally monitor the NN queries, only updates from objects falling in the expansion tree can alter the NN set of query. The expansion tree is based on a query point and used to facilitate handling of query movements while our AU structure is used to index the moving objects and predict their movement to reduce the index updates.

3 Data model and trajectory prediction

We use the GCA model we proposed in [7] to model the network and moving objects. A road network is modeled as a 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. GCA is based on the work of [21] and adapts the CA model for free traffic to represent the objects movement in road networks. 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 is treated as one CA connecting many cells.

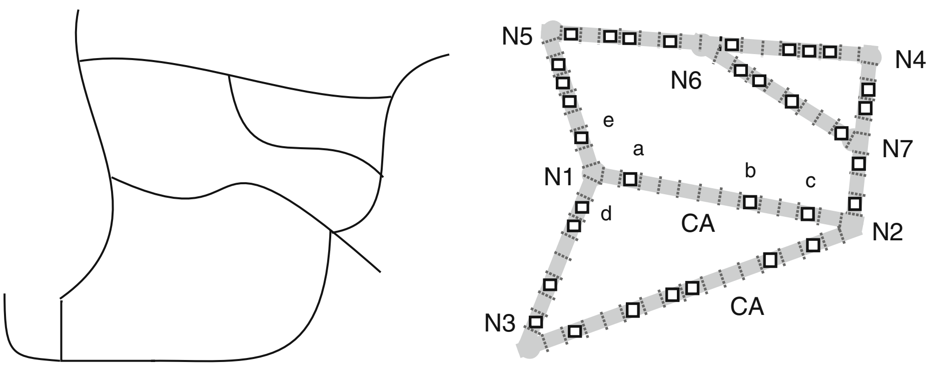
We first recall the definition of CA in this context.

Definition 1 A CA consists of a finite oriented sequence of cells. In a configuration, each cell is either empty or contains a symbol. During a transition, symbols can move forward to subsequent cells, symbols can leave the CA and new symbols can enter the CA.

An example of CA corresponding to edge (N_1, N_2) in Fig. 1b with a transition between two configurations is given in Fig. 2. We now formally define a GCA.

Definition 2 The structure of a GCA is a directed weighted graph $G = (V, E, l)$ where V is a set of vertices, E is a set of edges and $l : E \to \mathbb{N}$ is a function which associates to each edge the number of cells of the corresponding CA.

We assume a countably infinite alphabet $\Omega : \{a, b, c, \cdots\}$, denoting moving object's names. Cells are denoted by the edge name and their indices in the edge. Let C be the set of cells of a GCA. A configuration or an instance of a GCA, is a mapping from the cells of the GCA to constants in Ω together with a given velocity. Intuitively, the velocity is the number of cells an object can traverse during a time unit.



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

Fig. 1 An example of a road network and its GCA model

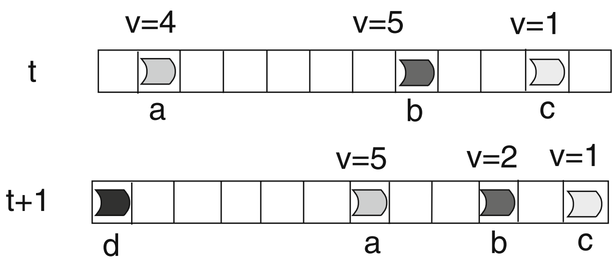


Fig. 2 Transition of the GCA

Definition 3 An instance I of a GCA is defined by two functions of the following types:
$\mu : C \rightarrow \Omega \bigcup \{\varepsilon\}$ (1-1 mapping)
$v : \Omega \rightarrow \mathbb{N}$.

A moving object is represented as a symbol attached to a cell in the GCA and it can move several cells ahead at each time unit. A moving object lies on exactly one cell of an edge and its location can be obtained by computing the number of cells relative to the starting node. For instance, object a lies on the edge (N_1, N_2) and there are two cells away from N_1 along the edge. Therefore, its position can be expressed by $(N_1, N_2, 2)$.

The motion of an object is represented as some (time, location) information. Representing such information of a moving object as a trajectory is a typical approach [35]. In the GCA model, the trajectory of a moving object can be divided into two types: the in-edge trajectory for the object's movement in one edge (CA) and the global trajectory for the object that may move cross several edges (CAs) during its movement. The in-edge trajectory of an object is a polyline in a two-dimensional space (one-dimensional relative distance, plus time), which can be defined as follows:

Definition 4 The in-edge trajectory of a moving object in a CA of length L is a piecewise function $f : T \rightarrow \mathbb{N}$, represented as a sequence of points $(t_1, l_1), (t_2, l_2), \ldots, (t_n, l_n)(t_1 < t_2 < \ldots < t_n, l_1 < l_2 < \ldots < l_n \leq L)$, where l_i is the relative distance to the starting node at the time of t_i.

When an object moves across multiple edges, its global trajectory is defined as functions mapping the time to the corresponding edge and the relative distance to the starting node.

Definition 5 The global trajectory of a moving object in different CAs is a piecewise function $f : T \rightarrow (E, \mathbb{N})$, represented as a sequence of points $(t_1, e_1, l_1), \ldots, (t_i, e_j, l_k), \ldots, (t_z, e_m, l_n)(t_1 < t_2 < \ldots < t_z)$.

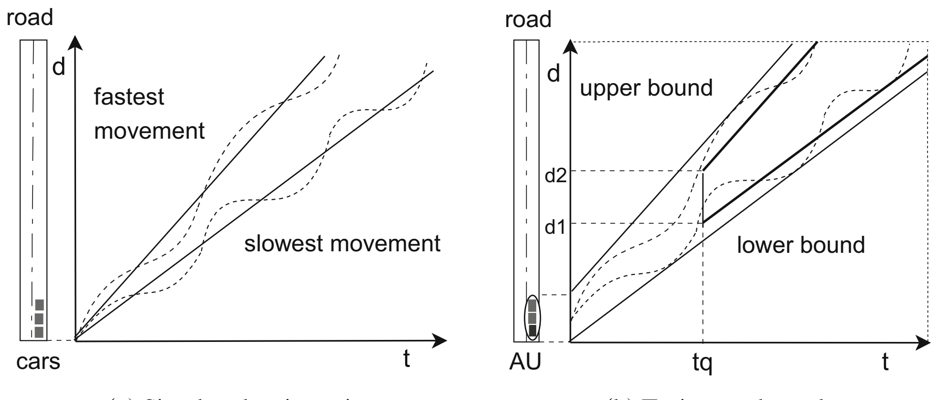
Let i be an object moving along an edge. Let $v(i)$ be its velocity, $x(i)$ its position, $gap(i)$ the number of empty cells ahead (forward gap), and $P_d(i)$ a randomized slowdown rate which specifies the probability it slows down. We assume that V_{max} is the maximum velocity of moving objects. The position and velocity of each object may change at each transition as shown in Definition 6.

Definition 6 At each transition of the GCA, each object changes velocity and position in a CA of length L according to the rules below:

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 an erratic movement behavior. Finally, the new position of object i is given by the fourth rule as sum of the previous position and the new velocity if the object is in the CA. Figure 2 shows the simulated movement of objects on a CA of the GCA in two consecutive timestamps. We can see that at time t, the speed of the object a is smaller than the gap (i.e. the number of cells between the object a and b). On the other hand, the object b will reduce its speed to the size of the gap. According to the fourth rule, the objects move to the corresponding positions based on their speeds at time $t + 1$.

We use GCAs not only to model road networks, but also simulate the movements of moving objects by the transitions of the GCA. Based on the GCA, the *SP* method to anticipate future trajectories of moving objects is proposed. The SP method treats the objects' simulated results as their predicted positions to obtain its future in-edge trajectory. To refine the accuracy, based on different assumptions on the traffic conditions we simulate two future trajectories in discrete points for each object on its edge. Then, by linear regression and translating, the trajectory bounds that contain all possible future positions of a moving object on that edge can be obtained. When the object moves to another edge in the GCA, another simulation and regression will be executed to predict new future trajectory bounds. The SP method is shown in Fig. 3.



(a) Simulated trajectories (b) Trajectory bounds

Fig. 3 The simulation-based prediction

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 a traffic system. However, we extend this model for predicting 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. 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. Our experiments show one of methods to choose the value of $P_d(i)$.

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 movements and are treated as one object.

4 The adaptive unit

4.1 Structure and storage

Conceptually, an AU is similar to a one-dimensional MBR in the TPR-tree, which 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 objects having similar moving patterns. Specifically, for objects in the same GCA edge, we use a distance threshold and a speed threshold to cluster the adjacent objects with the same direction and similar speed. The thresholds are set according to the average length of road segments, the average maximum speed on the segment and also the adaptation in the experimental data sets. In comparison, the AU has no obvious enlargement because objects in the AU move in a cluster.

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

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

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 predicted future trajectory of the AU. The trajectory bounds are derived from the trajectory bounds of the objects in the AU. We assume the functions of trajectory bounds as follows:

$$\texttt{upperBound}: \quad D(t) = \alpha_u + \beta_u \cdot t$$
$$\texttt{lowerBound}: \quad D(t) = \alpha_l + \beta_l \cdot t$$

`edgeID` denotes the GCA edge that the AU belongs to, `enterTime` and `exitTime` record the time when the AU enters and leaves the edge.

In the GCA, multiple AUs are associated with a GCA edge. Since AUs in the same edge are likely to be accessed together during query processing, we store AUs by clustering 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 a compact summary structure called the *direct access table* for each edge, which is treated as a secondary index of AUs can be accessed by a 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, objNum)`, where `auPtr` points to a list of AUs in disk storage and `objNum` 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 the simple network with small amount of AUs, we can keep all direct access tables in the main memory since it only keeps the summary information of AUs.

4.2 The index scheme

We build a spatial index (e.g., R-tree) for the GCA (road network) over the AUs to form the index scheme for the network-constrained moving objects. The AU index scheme 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. On 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 point to the lists of AUs. Each of 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 paper, but any existing spatial index can also be used without change.

Figure 4 shows the structure of the AU index scheme, which also includes a direct access table. The R-tree, the direct access table and AUs 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 AUs. 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 AUs 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.

4.3 Optimizing for updates

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 one-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

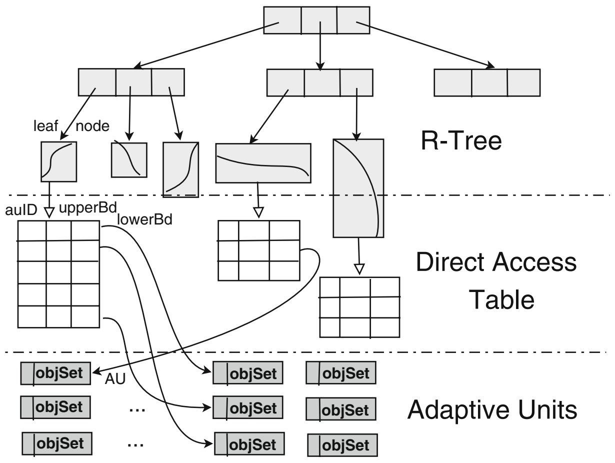


Fig. 4 Structure of the AU index scheme

in the future, the AU alleviates the update problem of MBR rapid expanding and overlaps in the TPR-tree like methods.

For reducing updates further, the AU captures the movement bounds of the objects based on the SP method, which considers the road-network constraints and stochastic traffic behavior. Since objects in an AU have similar movements, we then predict the movement of the AU by the SP method, as if it were a single moving object. The specific locations of the individual objects inside AUs can be similar and obtained by trajectory bounds of the AU. Through the SP method, we obtain two predicted future trajectory bounds of objects. When an object's position exceeds the AU, the index needs to be updated to delete the object from the old AU and insert the object to another AU. The accurate prediction of an AU's movement and expanding with objects' movement makes it possible that the updated location of each object seldom affects the changing of the AU structure (e.g. deleting and inserting objects, creating and dropping AUs). Therefore, the SP method helps to reduce the index updating costs.

The future trajectory bounds are predicted at each GCA node when an AU is created. The trajectory bounds will not be changed along the edge that the AU moves on until the objects in the AU move to another edge in the GCA. It is evident that the range of predicted bounds of an AU will become wider with the time, which leads to lower accuracy of future trajectory prediction. However, if we issue another prediction when the predicted bounds are not accurate any more, the costs of simulation and regression are high. Considering that the movement of objects along one GCA edge is stable, we can assume the same trends of the trajectory bounds and adjust only the initial locations when the prediction is not accurate. Specifically, the AU treats its actual locations (the locations of the boundary objects) at that time as the initial locations of the two trajectory bounds and follow the same movement vector (e.g. slope of the bounds) as the previous bounds to provide more accurate predicted trajectory bounds. In this way, the predicted trajectory bounds can be effectively revised with few costs. Figure 3b shows the adaptation of the trajectory bounds. t_q is the adaptation time of future trajectory bound and the d_1,d_2 are the actual locations of the first object and last object respectively in the AU. The trajectory bounds are revised according to the actual locations and the original bounds' slopes. Therefore, without executing more prediction, the prediction accuracy of the objects' future trajectories can be kept high.

Since the R-Tree indexes the GCA, it remains fixed, and the update of the AU index scheme restricts to the update of AUs. Specifically, an AU is usually created at the start of one edge and dropped at the end of the edge. Since the AU is a one-dimensional structure, it performs update operations much more efficiently than the two-dimensional indexes. We will describe these operations in details.

4.4 Update operations

The update of an AU can be of the following form: creating an AU, dropping an AU, adding objects to an AU and removing objects from an AU.

Creating an AU To create an AU, we first compose the *objSet* - 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). We then predict the future trajectories of the AU by simulation and compute 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. The prediction starts when the AU is created and ends at the end the edge. Finally, we write the created AU to the disk page and insert the AU entry to its summary structure. Factually, AU is created in two cases: 1) at the initial time with on bulk-loading at each network edge and 2) when the objects leave original edge with a single object described in Algorithm 1.

Dropping an AU When objects in an AU move out of the edge, they may change direction independently. So we need to drop this AU and create new AUs in adjacent edges to regroup the objects. When the front of an AU touches the end of the edge, some objects in the AU may start moving out of the edge. However, the AU cannot be dropped because a query may occur at that time. Only after the last object in the AU enters another edge and joins another AU, can the AU be dropped. Dropping an AU is simple. Through its entry in direct access table, we find the AU and delete it.

Adding and removing objects from an AU When an object leaves an AU, we 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. Specifically, when adding an object into an AU, we first find the direct access table of the edge that the object lies and, by its AU entry in the table, access the AU disk storage. Finally, we insert into the objects list of the AU and update the AU entry in the direct access table. Removing an object from an AU has the similar process.

Therefore, when updating an object in the AU index scheme, 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 to the object in terms of the network distance 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. When the AU is not updated, we check whether the object is the boundary object of the AU and whether its actual position exceeds the predicted bounds of the AU to a precision threshold ε (explained in the experiments of prediction accuracy), for the purpose of adapting the trajectory bounds of the AU. Factually, we find, from the experiment evaluation, that the chances that objects move beyond the trajectory bounds of its AU on an edge are very slim. The algorithm 1 shows the update algorithm when updating an object in the AU. Like the node capacity parameter in the index tree, MAXOBJNUM in the algorithm 1 is also used to restrict the number of object entries in an AU. It is set according to the object entry storage size and AU storage size.

In summary, updating the AU-based index is easier than updating the TPR-tree. It never invoke any complex node splitting and merging. Moreover, thanks 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.

4.5 Query algorithm

In this part, we propose an algorithm for predictive range query in the AU index scheme. It can also be extended to support the (K) Nearest Neighbor query

Algorithm 1 Update($objID$, $position$, $velocity$, $edgeID$)

```
input : $objID$ is the object identifier, $position$ and $velocity$ are its position and
        velocity, $edgeID$ is the edge identifier where the object lies
Find $AU$ where $objID$ is included before update;
if $AU.edgeID \neq edgeID$ or ($objID$ 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 $AU_1$ for $objID$ on $edgeID$;
    if GetNum($AU_1.objSet$) $< MAXOBJNUM$ and ObjectFitAU($objID$,
    $position$, $velocity$, $AU_1$)
    then
        InsertObject($objID$, $AU_1.auID$, $AU_1.edgeID$);
    else $AU_2 \leftarrow$ CreateAU($objID$,$edgeID$);
    if GetNum($AU.objSet$) $> 1$ then
        DeleteObject($objID$, $AU.auID$, $AU.edgeID$);
    else DropAU($AU.edgeID$, $AU.auID$);
else if ($objID$ is the low boundary object of $AU$ and $position$ exceeds
 $AU.lowerBound$ to $\varepsilon$) or ($objID$ is the high boundary object of $AU$ and
 $AU.upperBound$ exceeds $position$ to $\varepsilon$) then
    AdaptAUBounds($AU$, $position$);
```

and continuous query. A predictive range query captures all objects moving in a road network 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 AU index scheme consists of the following steps:

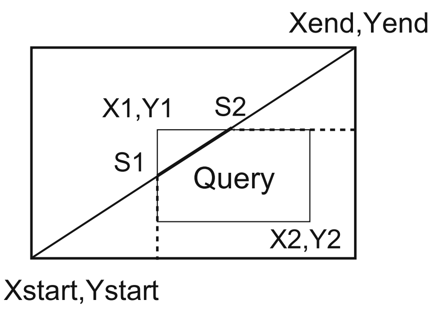
1) We first perform a spatial window range search (X_1, Y_1, X_2, Y_2) in the top level R-Tree to locate the edges (e.g. $e_1, e_2, e_3, \ldots$) that intersect the spatial query range.
2) For each selected edge e_i, we transform the original 3D search $(X_1, Y_1, X_2, Y_2, T_1, T_2)$ to a 2D search (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 5a gives an example when the query window range only intersects one edge e_1. 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 Fig. 5a and compute its relative distances S_1 and S_2. It can be easily extended to 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_1. According to Thales Theorem about similar triangles, we obtain S_1 and S_2 as follows:

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

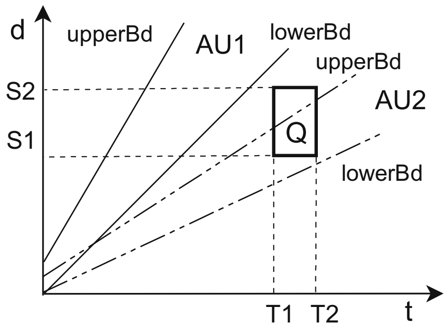
$$S_1 = \frac{X_1 - X_{start}}{X_{end} - X_{start}} r$$

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

3) We further find the adjacent edges of e_1 on which objects are possible to move into the window range during the future period $[T_1, T_2]$. For supporting future spatio-temporal range queries, the TPR-tree expands MBRs towards every direction according to the maximum speed of objects, which, when applied to the network, will result in large candidate result set including some objects that



(a) Query transformation

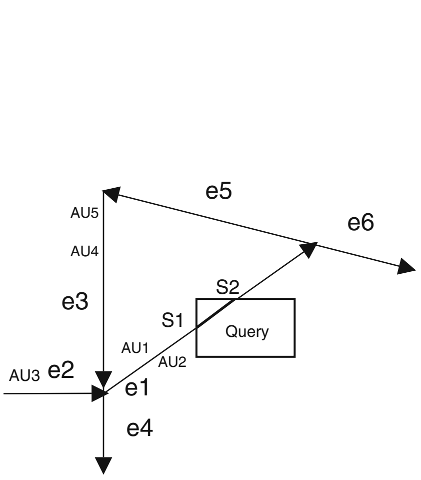


(b) Query process in AUs

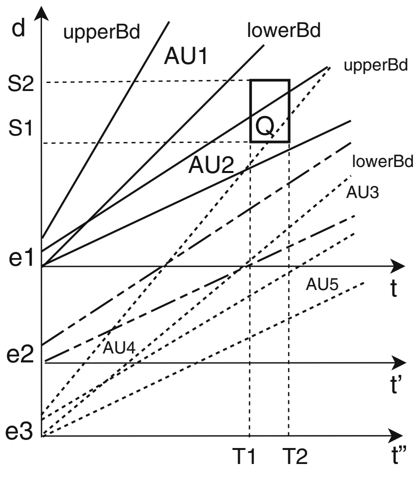
Fig. 5 Window range query in the AU index scheme

are impossible to move into the query range due to network constraint. There are limited possibilities of objects' movement in the road network. Therefore, we filter the candidate AUs in the adjacent edges possibly intersecting the window range by expanding along the network according to the maximum speed allowed in the network, adjacent table of edges and future query time. Figure 6 gives an example of network expanding in the query processing where arrow denotes the direction of edge. Let V_max the maximum speed and T_0 ($= 0$ in our example) the current time, it expands the network from the point of edge e_1 intersecting the spatial window (e.g. locations of S_1 in Fig. 6a) towards the reverse direction of e_1 and then continue to the adjacent edges obtained from the reverse adjacent table of e_1 until a expanded distance $V_{max} * (T_2 - T_0)$ is reached. The traversed edges e_2, e_3 in this example are returned. The AUs on these edges (e.g. AU3 on e_2 and AU4, AU5 on e_3 in Fig. 6a) will be further checked whether they are possible to intersect the query range during $[T_1, T_2]$. In this way, we can avoid the false negative for objects in the other edges 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_1. As illustrated by Fig. 5b, 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. The computations of transformed queries in adjacent edges e_2 and e_3 are also together showed in Fig. 6b. For the adjacent edge e_2 with the length of $l(e_2)$, we revise the transformed query to $(S_1 + l(e_2), S_2 + l(e_2), T_1, T_2)$ and filter AUs suitable to the query by linking e_2 and e_1, which is showed in the t'-d coordinate plane of Fig. 6b. We use the same method to filter AUs on the adjacent edge e_3 by linking e_3 and e_1, which is showed in the t''-d coordinate plane of Fig. 6b. This is reasonable to treat these AUs as candidates since the objects in them are also



(a) Network Expanding



(b) Query process in AUs

Fig. 6 Network expanding in query processing

likely to move to the query range in the future time. In our example, the query returns AU2 in edge e_1 and AU4 in adjacent edge e_3. 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.

Future spatio-temporal queries in a road network are more difficult to compute when considering the objects cross the different road segment edge because the future movement of objects in the road intersection is complex, but has limited possibilities due to the network constraint. We compute the query results which are likely to move to the road segment edge and location range in the near future that query intersects.

5 Performance analysis

In this section, we analyze the performance of the AU index scheme. We first analyze the I/O cost of the query and update algorithm, and then perform experimental evaluation.

5.1 Algorithms analysis

We follow the main assumptions of [33] 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 $N/(nM)$. We assume that B is the maximum number of objects in a disk page. The average number of AUs in a page is $ceiling(B/n)$.

For a spatio-temporal 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 [33]:

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

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

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

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 Fig. 3b, 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. Factually, μ_u and μ_l are the probabilities that a transformed point query is respectively above and below the trajectory bounds of the AU. 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 $\overline{\mu}_u$, $\overline{\mu}_l$ of the AUs on the edge using their bound functions and the length of the edge. Here we use the same analysis method for the computations of transformed queries in adjacent edges and ignore the cost of network expanding for simplicity. Now, we can get the average number of relevant AUs as follows:

$$\left(1 - \overline{\mu}_u^2 - \overline{\mu}_l^2\right) \frac{N}{nM}$$

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:

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

Therefore, the total I/O cost for a spatiotemporal window query in the AU index scheme is given by:

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

For improving the efficiency of the prediction of AU, the trajectory bounds of AU are computed based on the simulation not of all objects in it but of the object closest to the center of the AU. In this way, it seems that the query processing in the AU index will possibly not return correct query results (false negative) since the extrapolated position of the object at the query time will be outside of the bounds of the AU. However, this seldom happens for the following three reasons. 1) AU is constructed by a group of moving objects with similar moving pattern and maintained by tightly bounding enclosed moving objects for some time in the future. It is reasonable to approximate the AU by its center object. 2) In the SP method, to refine the prediction accuracy, we simulate two trajectories based on different assumptions on the traffic conditions (e.g. laminar and congested traffic) and translate the regressed lines outside to contain all possible future position of the object as soon as possible. 3) The adaptation of the trajectory bounds of AU also further improves the accuracy of trajectory prediction over time. From the experiments in Section 5.2.3, it is proved that such the approximation of AU simulation by its center object can achieve high efficiency improvement by causing very slim possibility of incorrect query results.

We then analyze the cost of updates in the AU index scheme. In order to update an object, we first scan AUs from the entries in the direct access table corresponding to its edge. The average number of disk accesses to find the AU page that the object belongs to (scan the pages of AUs on an edge) is $N/2MB$. If the object is entering a new edge or exceeds bounds of its AU, it will be inserted into the nearest AU and deleted from the old AU. This needs 1 disk accesses to read the nearest AU and 2 disk access to write the pages of the old AU and the new AU. If the object is the last object in the old AU, the old AU with the last object will be dropped which costs 1 disk access to write the page of the old AU. If the object cannot be inserted into the existing AU, a new AU will be created with 1 disk access. Therefore, the average cost of an update is $N/2MB + 3$.

5.2 Experimental evaluation

Since the R-tree in our structure only indexes the static spatial information of road networks, we compare it in the experiments with the TPR-tree-like method (taking the most popular TPR-tree for example) in which the R-tree is used to index the continuously changing moving objects. We measure the update performance with the individual update, update frequency and total update costs and the query performance of AU index scheme (denoted as “AU index”), the TPR-tree and the AU index scheme when the direct access table is not used (denoted as “AU index without DT”). We then study the effect of parameter P_d on the SP and finally compare the prediction accuracy of the SP method with that of the linear prediction method.

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’s Network-based Generator [5]. 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 time-stamp. The Brinkhoff’s 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 time-stamp. We implemented both the AU index scheme and the TPR-tree in Java and carried out experiments on a Pentium 4, 2.4 GHz PC with 512 MB RAM running Windows XP. To improve the performance of the index structure, we employed a LRU buffer of the same size (200K) as the one used in the TPR-tree [29]. Especially, for the AU index with DT, the LRU buffer is used for the R-tree nodes, AU pages, and DAT pages. The same amount of main memory is allocated to buffers for all of the three compared index structure. Since the map of Oldenburg is a relatively small network and the direct access tables are frequently accessed, most of them are kept in the main memory in our experiments.

Table 1 Parameters and their settings

Parameters	Settings
Page size	4K
Node capacity	100
Numbers of queries	200
Numbers of mo(cars)	10K, ... , 50K, ... , 100K
Numbers of updates	100K, ... , 500K, ... , 1M
Dataset generator	CA simulator, network-based generator

We summarize workload parameters in Table 1, where values in bold are default values.

5.2.2 Update cost

We compare the cost of index update for the AU index and the TPR-tree in terms of the average individual update cost, update frequency and total update cost.

Individual Update Cost We study the individual update performance of the index while varying the number of moving objects and updates. Figure 7 shows the average individual update cost when increasing the data size from 10K to 100K. Figure 8 shows how the performance varies over time. Clearly, updating the TPR-tree tends to be costly, and the problem is exacerbated when the data size increases. In each case of different data size and different number of updates, the AU index has much lower update cost than the TPR-tree. The main reason can be explained as follows. Each update of the TPR-tree involves the search of an old entry and a new entry, as well as the modification of the index structure (node splitting, merging, and the propagating of changes upwards). The cost increases with larger data size due to more overlaps among MBRs. The changes of index structure with the increase of data updates also affect the performance of the TPR-tree. However, the AU index has better performance because its update only restricts to the AU's update and as a one-dimensional access structure, the AU has few overlaps and incurs no cost associated with node splitting and the propagation of MBR updates.

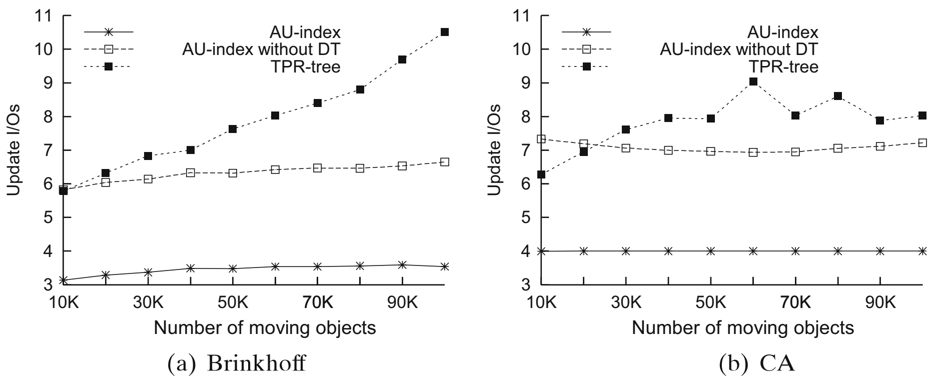


Fig. 7 Individual update cost with different datasize

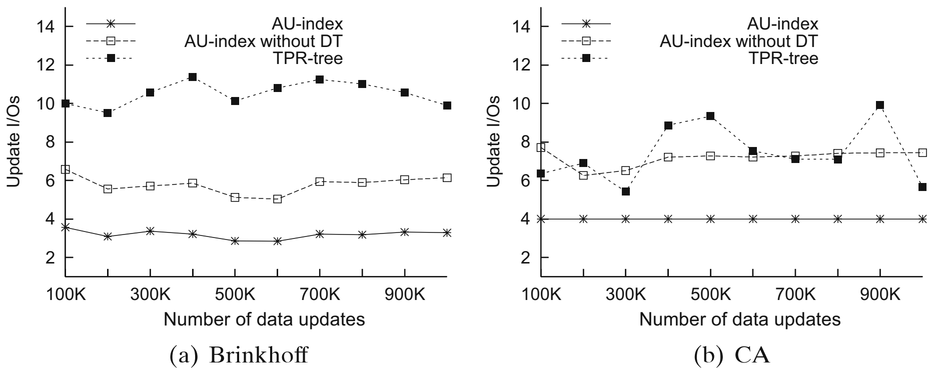


Fig. 8 Individual update cost over time

The direct access table of the AU index has a significant contribution in improving update performance. This is because searching the specific AU is accelerated by the secondary index structure.

Update Frequency Frequent updates of moving objects (a.k.a. data updates) may lead to frequent updates of index. All data updates (that are received according to some object update policy) should be recorded in the index, but may lead to different numbers of "index update" for the AU and TPR-tree respectively. In our experimental evaluations, "index update" for AU denotes an update of an object that invalidates the objects AU. For example, when an object's position exceeds the bounds of AU, the index needs to be updated to delete the object from the old AU and insert it to another one. For the TPR-tree, the bounding rectangles are recomputed even if they are not invalidated. We did not count such re-computations as "index update" and only counted the updates which invalidated the bounding rectangles of TPR-tree. In this experiment, we measure the "index update rate", which is the ratio between number of such index update and number of data update, for every 100K data updates and different data size. Figures 9 and 10 show that the

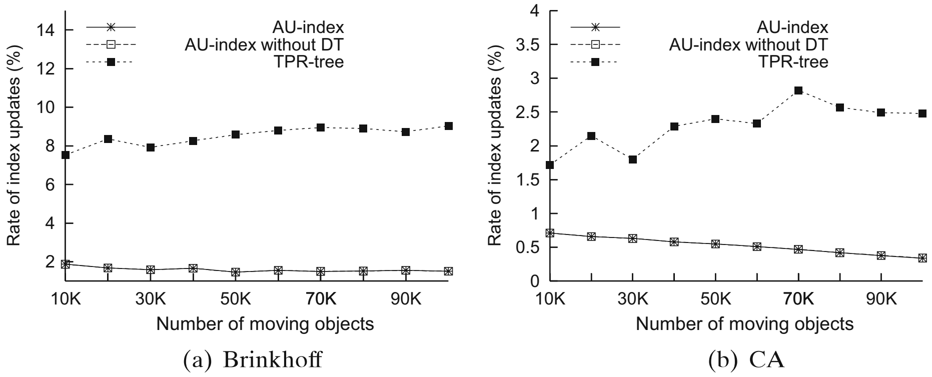


Fig. 9 Index update frequency with different datasize

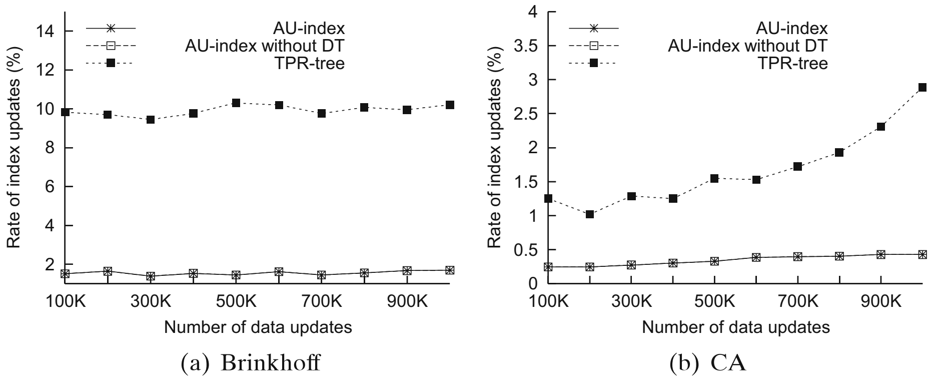


Fig. 10 Index update frequency over time

update rate of the TPR-tree is nearly 4 to 5 times more than that of the AU index. The index update rate depends on the prediction method. In the AU index, the future positions of the object are predicted more accurately, so the object is likely to remain in its AU, which leads to fewer index updates. As to the reduction in the object update frequency when the SP method is used, we also evaluate the performance in paper [8].

Total Update Costs The total update costs depend on the update frequency and the average individual update cost, and it can reflect the index update performance more accurately. From both Figs. 11 and 12, we can see that although the AU index has to deal with the creation and dropping of AUs, the TPR-tree incurs much higher update costs than the AU index 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).

For each data size, the update costs of the two indexes in the Brinkhoff's 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's dataset.

5.2.3 Query cost

Effect of Data Size We study the window range query performance of the TPR-tree and the AU index while varying the number of moving objects from 10k to 100k. Figure 13 shows the average number of I/O per query with query window size 50. In each case, the query cost increases as the data size increases. However, the AU index has much lower cost than the TPR-tree. This is because the AUs in the AU index have much less overlaps than the MBRs in the TPR-tree, and the overlaps to a large extent determine the range query cost.

The AU index with the direct access table achieves better performance than the AU index without it. This is because the secondary index structure enables us to filter some unnecessary AUs during the search of AUs that intersect the range query. However, for the Brinkhoff's dataset the benefit of the direct access table is not obvious because the large number of small edges in the network reduces chances of

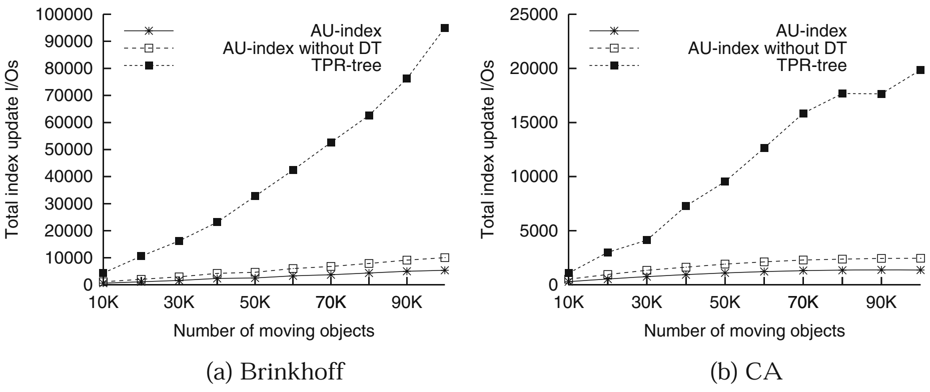


Fig. 11 Total update cost with different datasize

filtering the AUs not included in the range query. The search costs of the two indices in different datasets are also quite different for the same reasons as mentioned in update performance.

Effect of Update We then study the window range query performance of the TPR-tree and the AU index with different update settings. We increase the number of updates from 100K to 1M to examine how query performance is affected. We issued 200 range queries with window size 50 for every 100K updates in a 1M dataset. Figure 14 shows that the cost of the TPR-tree increases much faster as the number of updates increases. The cost of the AU index 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 AU index, the increase of the updates hardly affect the total number of AUs, and the chances of overlaps of different AUs are very slim.

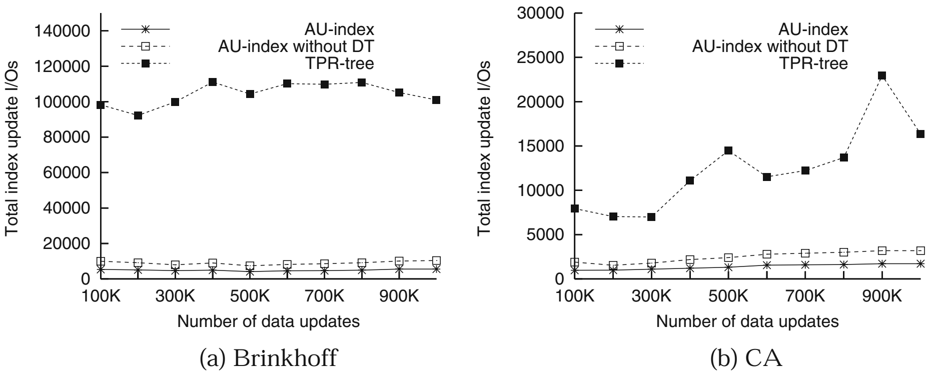


Fig. 12 Total update cost over time

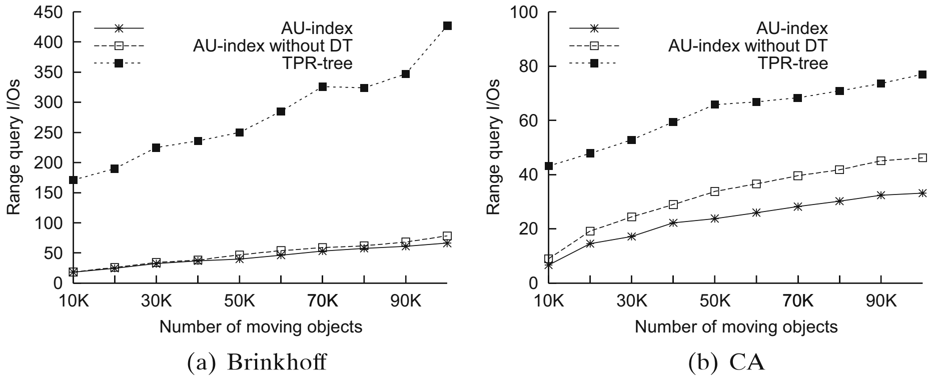


Fig. 13 Query performance with data size

Effect of Query Window Size To study the effect of query window size on performance, we increase the window size from 10 to 100 (the fraction of the total space from 1/1000 to 1/100) for 100K data size with a workload of 200 range queries. Figure 15 shows the query cost as a function of the query window size. It is clear that for all the indexes, 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.

Query Recall With the same dataset and the window size 50, we measure the query recall in AU index with the approximation of AU simulation by its center object. By referencing the query results in the TPR-tree, we compute the false negative and compare the efficiency of the AU index with approximation and without approximation. The results show that the approximation of AU simulation by its center object can achieve high efficiency improvement (average 25% increase of efficiency) by causing very slim possibility (average 5% possibility) of incorrect query results (average 6% false negative).

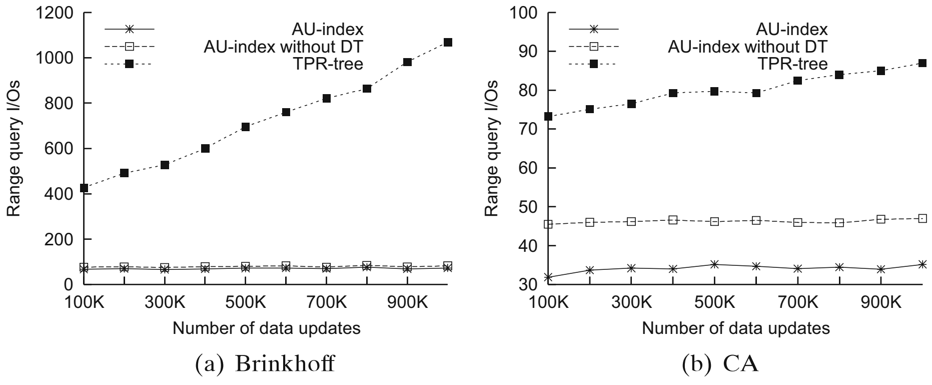


Fig. 14 Effect of updates on query

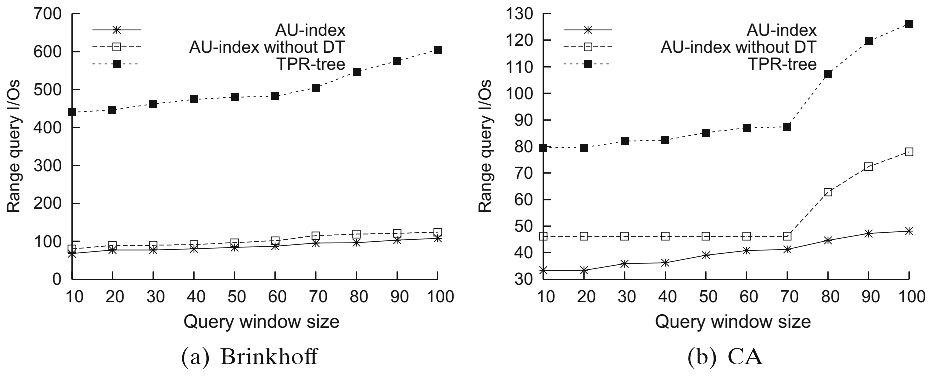


Fig. 15 Effect of query window size on query performance

5.2.4 Prediction accuracy

The SP method can be used in the AU index structure to reduce the indexing updates cost in that it is more accurate in predicting the future trajectories of AU with some similar objects than the linear prediction method. For simplicity, we study the effect of simulation parameter and evaluate the prediction accuracy by applying prediction method to the location update policy of object [8].

The Slowdown Rate P_d The simulation has an important effect on the prediction accuracy and therefore affects the efficiency of query and update. We study the effect of the choice of different P_d, which determines the two predicted trajectories corresponding to the fastest and slowest movement. We test on the Brinkhoff's dataset with different data size and use P_d from 0 to 0.5 and measure the average prediction accuracy by "average error" and "overflow rate". The average error is the average absolute error between the predicted and actual positions, and the overflow rate represents the probability of predicted positions exceeding the actual positions. The purpose of this metric is to find the closest two trajectories binding the actual one

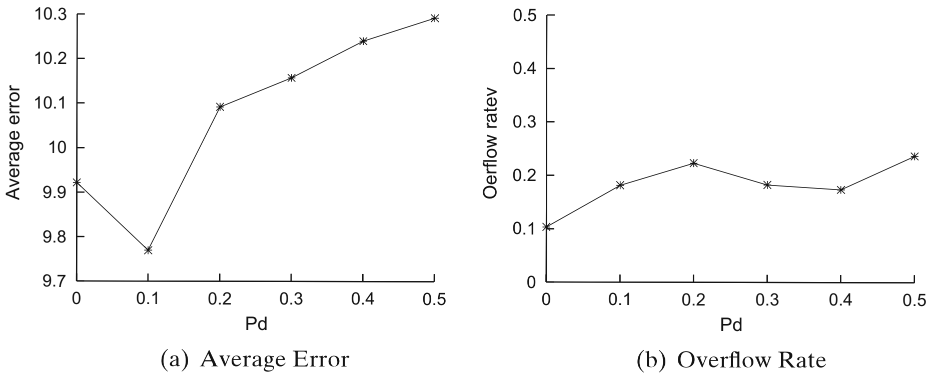


Fig. 16 Prediction accuracy with P_d

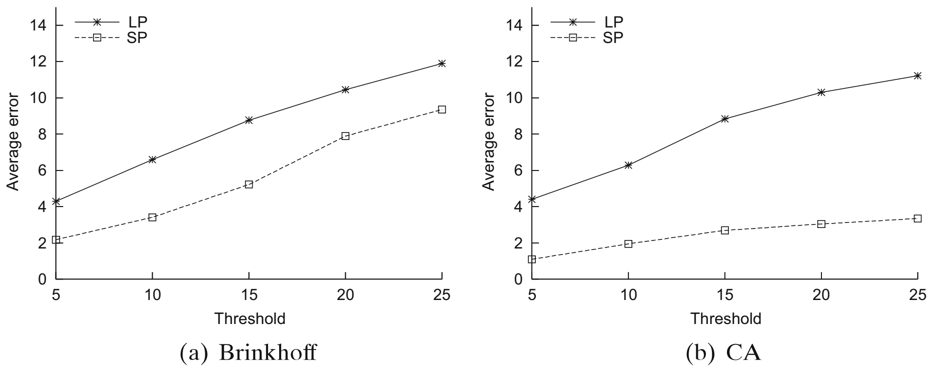


Fig. 17 Prediction accuracy with threshold

as future trajectories. In this way, we can choose the P_d both with lower average error and overflow rate. Figure 16 shows the prediction accuracy of the SP with different slowdown rates. We can see that when P_d is set to 0 and 0.1, both the average error and overflow rate are lower than others. Therefore, we use the value 0 and 0.1 as slowdown rates for the fastest movement bound and the slowest movement bound to obtain better prediction results.

Prediction Accuracy and Cost Finally, we compare the precision of the SP method with the LP method. We measure the prediction accuracy by "average error" but with different threshold ε. The threshold ε represents the maximum deviation between the predicted locations of an object and its real locations allowed in the prediction. That means when the deviation exceeds the threshold ε, we make another prediction. From Fig. 17, we observe that average error will increase when threshold increases. This is because the larger the threshold is, the larger the deviation becomes, which leads to the more errors. It is tenable in both the LP and SP method. However, the SP method predicts more accurately than the LP method with any threshold ε.

To measure the time cost of the prediction, we compute the average CPU time when simulating and predicting the movement of one object along the edge with length 1000 in different dataset sizes. The results show that the average cost of one SP is about 0.25 ms. This is quite acceptable.

6 Conclusions and future work

Indexing objects moving in a constrained network especially the road network is a topic of great practical importance. We focus on the index update issue for the current positions of network-constrained moving objects. We introduce a new access structure, AU that exploits as much as possible the characteristics of the movements of objects. The updates of the structure are minimized by an accurate prediction method which produces two trajectory bounds based on different assumptions on the traffic conditions. The efficiency of the structure also results from the possible reduction of dimensionality of the trajectory data to be indexed. Our experimental results performed on two datasets show that the efficiency of the index structure is one order of magnitude higher than the TPR-tree.

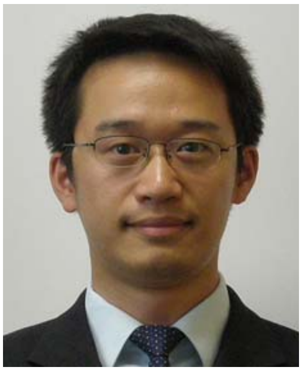
We will extend the query algorithms to support the KNN query and continuous query for moving objects in the road network.

Acknowledgements This research was partially supported by the grants from the Natural Science Foundation of China under grant number 60573091; China 863 High-Tech Program (No. 2007AA01Z155); China National Basic Research and Development Program's Semantic Grid Project (No. 2003CB317000); Program for New Century Excellent Talents in University (NCET). Program for Creative PhD Thesis in University. The authors would like to thank Jianliang Xu and Haibo Hu from Hong Kong Baptist University and Stéphane Grumbach from CNRS, LIAMA in China for many helpful advice and assistance. The author also appreciates Yanyan, Guo and Zhen Xiao from Renmin University of China for helping the implementation of the experiments and Wenbo, Mao from EMC Research China for English improvement.

References

1. Aggarwal C, Agrawal D (2003) On nearest neighbor indexing of nonlinear trajectories. In: Proc. of the 22nd ACM SIGMOD-SIGACT-SIGART symp. on principles of database systems, San Diego, 9–11 June 2003, pp 252–259
2. Agarwal PK, Arge L, Erickson J (2000) Indexing moving points. In: Proc. of the 19th ACM SIGMOD-SIGACT-SIGART symp. on principles of database systems, Dallas, 15–18 May 2000, pp 175–186
3. Almeida VT, Güting RH (2005) Indexing the trajectories of moving objects in networks. Geoinformatica 9(1):33–60
4. Beckmann N, Kriegel HP, Schneider R, Seeger B (1990) The R*-tree: an efficient and robust access method for points and rectangles. In: Proc. of the ACM SIGMOD int. conf. on management of data, Atlantic City, May 1990, pp 322–331
5. Brinkhoff T (2002) A framework for generating network-based moving objects. Geoinformatica 6(2):153–180
6. Cho H, Chung C (2005) An efficient and scalable approach to CNN queries in a road network. In: Proc. of the 31st int. conf. on very large data bases, Trondheim, 30 August–2 September 2005, pp 865–876
7. Chen J, Meng X, Guo Y, Grumbach S, Sun H (2006) Modeling and predicting future trajectories of moving objects in a constrained network. In: Proc. of the 7th int. conf. on mobile data management (MDM), Nara, 9–13 May 2006, pp 156 (MLASN workshop)
8. Chen J, Meng X, Li B, Lai C (2006) Tracking network-constrained moving objects with group updates. In: Proc. of the 7th int. conf. on web-age information management (WAIM), Hong Kong, 17–19 June 2006, pp 158–169
9. Cheng R, Xia Y, Prabhakar S, Shah R (2005) Change tolerant indexing for constantly evolving data. In: Proc. of 21st int. conf. on data engineering, Tokyo, 5–8 April 2005, pp 391–402
10. Ding Z, Güting RH (2004) Managing moving objects on dynamic transportation networks. In: Proc. of 16th int. conf. on scientific and statistical database management (SSDBM), Santorini Island, 21–23 June 2004, pp 287–296
11. Frentzos E (2003) Indexing objects moving on fixed networks. In: Proc. of the 8th intl. symp. on spatial and temporal databases, Santorini Island, July 2003, pp 289–305
12. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proc. of the ACM SIGMOD int. conf. on management of data, Boston, June 1984, pp 47–57
13. Jensen CS, Kollios J, Pedersen TB, Timko I (2003) Nearest neighbor queries in road networks. In: Proc. of the 11th ACM int. symp. on advances in geographic information systems, New Orleans, 7–8 November 2003, pp 1–8
14. Jensen CS, Lin D, Ooi BC (2004) Query and update efficient B+-tree based indexing of moving objects. In: Proc. of 30th int. conf. on very large data bases, Toronto, 29 August–3 September 2004, pp 768–779
15. Kolahdouzan MR, Shahabi C (2004) Voronoi-based K nearest neighbor search for spatial network databases. In: Proc. of 30th int. conf. on very large data bases, Toronto, 29 August–3 September 2004, pp 840–851

16. Kollios G, Gunopulos D, Tsotras VJ (1999) On indexing mobile objects. In: Proc. of the 8th ACM SIGMOD-SIGACT-SIGART symp. on principles of database systems, Philadephia, 31 May–2 June 1999, pp 261–272
17. Kim K, Kim S, Kim T, Li K (2003) Fast indexing and updating method for moving objects on road networks. In: Proc. of 4th int. conf. on web information systems engineering, Los Alamitos, 10–12 December 2003, pp 34–42
18. Kwon D, Lee SJ, Lee S (2002) Indexing the current positions of moving objects using the lazy update R-tree. In: Proc. of the 3rd int. conf. on mobile data management, Singapore, 8–11 January 2002, pp 113–120
19. Lee ML, Hsu W, Jensen CS, Cui B, Teo KL (2003) Supporting frequent updates in R-trees: a bottom-up approach. In: Proc. of 29th int. conf. on very large data bases, Berlin, 9–12 September 2003, pp 608–619
20. Mouratidis K, Yiu ML, Papadias D, Mamoulis N (2006) Continuous nearest neighbor monitoring in road networks. In: Proc. of 32nd int. conf. on very large data bases, Seoul, 12–15 September 2006, pp 43–54
21. Nagel K, Schreckenberg M (1992) A cellular automaton model for freeway traffic. J Physique 2:2221–2229
22. Nascimento MA, Silva JRO (1998) Towards historical R-trees. In: ACM symposium on applied computing, Atlanta, 27 February–1 March 1998, pp 235–240
23. Patel JM, Chen Y, Chakka VP (2004) STRIPES: an efficient index for predicted trajectories. In: Proc. of the ACM SIGMOD int. conf. on management of data, Paris, 15–17 June 2004, pp 637–646
24. Pfoser D, Jensen CS, Theodoridis Y (2000) Novel approaches in query processing for moving object trajectories. In: Proc. of 26th int. conf. on very large data bases, Cairo, 10–14 September 2000, pp 395–406
25. Pfoser D, Jensen CS (2003) Indexing of network constrained moving objects. In: Proc. of 11th ACM int. symp. on advances in geographic information systems, New Orleans, 7–8 November 2003, pp 25–32
26. Papadias D, Zhang J, Mamoulis N, Tao Y (2003) Query processing in spatial network databases. In: Proc. of the 29th int. conf. on very large data bases (VLDB), San Fransisco, May 2003, pp 790–801
27. Saltenis S, Jensen CS (2002) Indexing of moving objects for location-based service. In: Proc. of 18th int. conf. on data engineering, San Jose, 26 February–1 March 2002, pp 463–42
28. Speicys L, Jensen CS, Kligys A (2003) Computational data modeling for network-constrained moving objects. In: Proc. of the 11th ACM int. symp. on advances in geographic information systems, New Orleans, 7–8 November 2003, pp 118–125
29. Saltenis S, Jensen CS, Leutenegger ST, Lopez MA (2000) Indexing the positions of continuously moving objects. In: Proc. of the ACM SIGMOD int. conf. on management of data, Dallas, 16–18 May 2000, pp 331–342
30. Shababi C, Kolahdouzan MR, Sharifzadeh M (2003) A road network embedding technique for K-nearest neighbor search in moving objects databases. Geoinformatica 7(3): 255–273
31. Tao Y, Papadias D (2001) The MV3R-tree: a spatiotemporal access method for timestamp and interval queries. In: Proc. of 27th int. conf. on very large data bases, Roma, 11–14 September 2001, pp 431–440
32. Tao Y, Papadias D, Sun J (2003) The TPR*-tree: an optimized spatiotemporal access method for predictive queries. In: Proc. of 29th int. conf. on very large data bases, Berlin, 9–12 September 2003, pp 790–801
33. Theodoridis Y, Stefanakis E, Sellis TK (2000) Efficient cost models for spatial queries using R-trees. IEEE Trans Knowl Data Eng 12(1):19–32
34. Tao Y, Faloutsos C, Papadias D, Liu B (2004) Prediction and indexing of moving objects with unknown motion patterns. In: Proc. of the ACM SIGMOD int. conf. on management of data, Paris, 15–17 June 2004, pp 611–622
35. Vazirgiannis M, Wolfson O (2001) A spatiotemporal model and language for moving objects on road networks. In Proc. of 7th int. sym. on spatial and temporal databases (SSTD), Redondo, 12–15 July 2001, pp 20–35
36. Xiong X, Aref WG (2006) R-trees with update memos. In: Proc. of 22nd int. conf. on data engineering, Atlanta, 3–7 April 2006, pp 22
37. Yiu ML, Tao Y, Mamoulis N (2008) The $B^{dual} - tree$: indexing moving objects by space-filling curves in the dual space. VLDB 17(3):379–400



Chen Jidong is a Research Scientist at EMC Research China, one of the research groups at EMC Corporation. He received his bachelor and master from Southwest Petroleum University, and his Ph.D. degree from Renmin University of China, all in computer science. His research interests include personal information management, collaborative knowledge management and mobile information management. He has published over 15 papers in refereed international journals and conference proceedings.

He has held visiting research positions at the PRISM Lab of Versailles University in Paris (2005) and Hong Kong Baptist University (2006). He has served as the local organizing committee of MDM 2008 conference and as a reviewer to journal of Future Generation Computer Systems. He will also serve as the program committee for EDBT 2009 industrial and applications track.

Meng Xiaofeng is a full professor at School of Information, Renmin University of China. He received a B.S. degree from Hebei University, M.S. degree from Remin University of China , Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, all in computer science. He is currently the Vice Dean of School of Information and the Head of Department of Computer Science. He is the Secretary General of Database Society of the China Computer Federation (CCF DBS), which is the largest national academic organization in China established in 1977. He has held visiting research position at the Chinese University of Hong Kong (1994,1995,1998), City University of Hong Kong (1997), visiting scientist at the National University of Singapore (2003), and visiting professor at Prism Lab of Versailles University, Paris(2004).

His research interests include query processing, natural language interface, web data extraction, native XML databases, mobile data management. He has published over 100 papers in refereed international journals and conference proceedings.

He has served on the program committee SIGMOM, ICDE, ER, DASFAA, MDM, DEXA, WISE, WAIM, etc. and as a reviewer to journals including IEEE Transactions on Knowledge and Data Engineering.