Efficient Indexing of the Past, Present and Future Positions of Moving Objects on Road Network

Ying Fang[1], Jiaheng Cao[1], Yuwei Peng[1], Nengcheng Chen[2], and Lin Liu[2]

[1] School of Computer, Wuhan University, China
{fangying,jhcao,ywpeng}@whu.edu.cn
[2] State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, China
cnc_dhy@hotmail.com

Abstract. Aim at moving objects on road network, we propose a novel indexing named PPFN*-tree to store past trajectories, present positions, and predict near future positions of moving objects. PPFN*-tree is a hybrid indexing structure which consists of a 2D R*-tree managing the road networks, a set of TB*-tree indexing objects' movement history trajectory along the polylines, and a set of basic HTPR*-tree indexing the position of moving objects after recent update. PPFN*-tree can not only support past trajectory query and present position query, but also support future predictive query. According to the range query time, query in PPFN*-tree can be implemented only in the TB*-tree, or only in the HTPR*-tree, or both of them. Experimental results show that the update performance of the PPFN*-tree is better than that of the PPFI and the R^{PPF}-tree. The query performance of the PPFN*-tree is better than that of the MON-Tree and the PPFI.

Keywords: moving object indexing, PPFN*-tree, TB*-tree, HTPR*-tree, range query, trajectory query.

1 Introduction

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

Numerous researches are focused on index structures for moving objects. They can be classified into two major categories depending on whether they deal with past information retrieval or future prediction. However, some queries not only involved past positions but also current and future positions of moving objects. Many researchers begin to focus on the index structure of moving object for support querying about the past, the present and the future.

Most of these approaches for indexing moving objects assume free movement of objects in the 2-dimensional space. According to reference [1], applications dealing

Y. Gao et al. (Eds.): WAIM 2013 Workshops, LNCS 7901, pp. 223–235, 2013.

with moving objects can be grouped into three movement scenarios, namely unconstrained movement, constrained movement, and movement in transportation networks. The latter category is an abstraction of constrained movement, i.e., for cars, one might be only interested in their position with respect to the road network, rather than in absolute coordinates. Then, the movement can be viewed as occurring in a different space than for the first two scenarios, which is called 1.5 dimensional spaces in [2].

For the constrained movement scenario, most index structures are focused on indexing the history trajectories or current position of moving objects. For example, index method proposed by C. S. Jensen *et al.* [3], the Fixed Network R-Tree [4] and MON-tree [5] store the complete trajectories of the moving objects and are capable to answer queries about the past states of the database. However, all of these works are focused on the historical movement and cannot support queries related to the current positions and near future positions of moving objects. IMORS [6] is a hybrid index method for indexing current positions of moving objects on road sectors. It relieves update overheads and provides efficient mechanisms in both searching and updating. However, only current positions of the moving objects can be searched in IMORS.

PPFI [7] is a hybrid indexing structure which consists of a 2D R*-tree built on polylines describing road sectors for managing the fixed networks, a set of 1D R*-Trees indexing objects' movement along the polylines, and a hash structure describing the recent state of moving objects. It is capable of answering query related to the past and current positions of moving objects, and predicting near future positions of moving objects. However, some queries such as spatio-temporal range queries related to position information after the most recent update could not be realized in PPFI.

In this paper, we address the problem of indexing moving objects on road network from the past to the future. We propose a novel indexing method named PPFN*-tree (Past-Present-Future index of Moving Object on Road Network) to store past trajectories, present positions, and predict future positions of moving objects on road network. PPFN*-tree is a hybrid indexing structure that consists of a 2D R*-tree which is built on polylines describing road sectors for managing the road networks, a set of TB*-trees indexing objects' movement history trajectory along the polylines, and a set of basic HTPR*-tree indexing the position of moving objects after recent update. In addition, two hash tables are included in PPFN*-tree. PPFN*-tree can not only support past trajectory and present position query, but also support future predictive query.

The organization of this paper is as follows. Section 2 presents related works and motivations of our work. Section 3 describes data model and update policy in PPFN*-tree, shows the index structure of PPFN*-tree and the corresponding algorithms. Section 4 reports on the performance evaluation. The conclusion is given in section 5.

2 Related Works and Motivations

A number of index structures have been proposed for moving object database. Most of these index structures are focused on free movement of the objects in the 2-dimensional

space. Some index structures only handles past positions or trajectories of moving object focused on free movement in space. The STR-tree [8] attempts to group segments according to their trajectory memberships, also taking spatial locations into account. The TB-tree [8] aims only for trajectory preservation, leaving other spatial properties aside. Based on the MVB-tree, Tao and Papadias propose the MV3R-tree [9] which consists of an MVR-tree and a 3D R-tree to index past trajectory data. Lee Eung Jae *et al.* propose TB* tree [10] for efficiently managing current and past trajectory of moving objects. The proposed method considerably improves updating performance using *auxiliary cache* and reduces index size by removing redundant data for representing MBB. However, all the above indices capture only the positions of objects from some past time up until the time of the most recent update.

Some structures only index the current and near-future positions of moving objects. For example, Tayeb et al. [11] use the PMR-quadtree as their underlying spatial access methods for indexing the future trajectories. Kollios et al. [12] and Papadopoulos et al. [13] use the duality transformation to transform a line segment (e.g., trajectory) from the time-space domain into a point in the two-dimensional space. By introducing parametric bounding rectangles in R-tree, the TPR-tree [14] provides the capability to answer the queries about current positions and future positions. The TPR*-Tree [15] improved upon the TPR-Tree by introducing a new set of penalty functions based on a revised query cost model. Based on the B^+-tree, indices for moving objects not only supporting queries efficiently but also supporting frequent updates are proposed. Jensen et al. propose the B^x-tree [16], which employs space partitioning and data/query transformations to index object positions and has good update performance. ST^2B-tree [17] is a Self-Tunable Spatio-Temporal B+-Tree index for moving object database, which is amenable to tuning. Based on the TPR*-tree, basic HTPR*-tree [18][19] adds creation or update time of moving objects to leaf node entries. Basic HTPR*-tree not only supports predictive queries, but also supports partial history queries involved from the most recent update instant of each object (t^o_{mru}) to the last update time (t_{lu}) of all objects.

Some indexing methods are proposed to support both the past and the future movement of the objects. Sun et al. [20] propose a method for approximate query answering based on multidimensional histograms. The BB^x-tree proposed by Lin et al. [21] retains the old phases so that past, present, and future positions of moving objects are indexed, but it only indexes broken "polylines". Based on the TPR-tree, Pelanis et al. [22] propose the R^{PPF}-tree. It can not only accurately index position for times in-between the most recent instant of each object and the last update time of all objects, but also describe connected trajectories of objects. PPFI*[23] is a hybrid indexing structure which consists of a TB-tree indexing history trajectories from some past time until the time of the most recent position sample of each object $o(t^o_{mru})$, and a HTPR*-tree describing position information since the most recent update instant of each object (t^o_{mru}). It not only supports queries of the positions of moving objects at all points in time, but also supports frequent update.

In order to manage moving objects on road network, some special index structures have been developed. For example, C. S. Jensen et al. [3] proposed an index method which stores the network edges as line segments in a 2D R-Tree and the moving objects

in another 2D R-Tree to index the past trajectories of moving objects in networks. FNR-Tree [4] separates spatial and temporal components of the trajectories and indexes the time intervals that each moving object spends on a given network link. The MON-tree approach [5] further improves the performance of the FNR-tree by representing each edge by multiple line segments instead of just one. However, all of these works are focused on the historical movement of moving objects, and query related to the current positions and near future positions of moving objects could not be realized in these index structures.

IMORS [6] is a hybrid index method for indexing current positions of moving objects on road sectors. It relieves update overheads and provides efficient mechanisms in both searching and updating. However, only current positions of the moving objects can be searched in IMORS. PPFI [7] is a hybrid indexing structure which consists of a 2D R*-tree built on polylines describing road sectors for managing the fixed networks, a set of 1D R*-trees indexing objects' movement along the polylines, and a hash structure describing the recent state of moving objects. It is capable of answering query related to the past and current positions of moving objects, and predicting near future positions of moving objects. Because PPFI uses a hash structure describing the recent state of moving objects, it only supports the prediction of near future positions of moving objects and does not provide predictive range query and predictive nearest neighbor (NN) query.

Our work aims to provide an indexing structure not only to support frequent update of moving objects positions, but also to provide querying from the past to the future for moving objects on road networks.

3 PPFN*-Tree

3.1 Data Modeling and Update Policy

As for object moving on road network, road network model is necessary. In PPFN*-tree, the road network is represented in terms of routes and junctions between the routes, i.e., a network $G=(R, J)$, where R is a set of routes and J is a set of junctions. A route $r \in R$ has an associated polyline $pl = p_0, \ldots, p_n$, where p_i are 2-dimensional points, $0 \leq i \leq n$, and $n+1$ is the size of the route. We term p_0 the start point and p_n the end point of the polyline. The direction of a polyline is from its start point to its end point. A polyline, exemplified in Figure 1, can be described as a sequence of connected line segments.

The measure (m) of a point p located on a polyline is the distance, measured along the polyline, from the start point to point p; see Figure 1.

Function M calculates the measure of any point p located on a polyline pl.

$$M(pl, p) = \begin{cases} 0 & p = p_0 \\ M(pl, p_i) + d(p_i, p) & \exists i (i = \min(\{ j \mid p \in s_j \wedge s_j \in pl) \\ \text{undefined} & \text{otherwise} \end{cases}$$

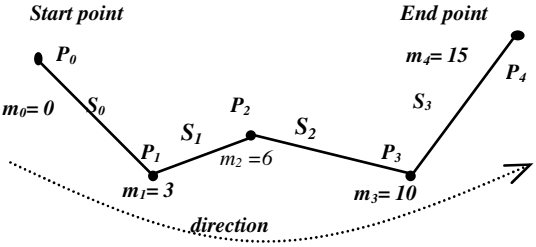


Fig. 1. Polyline

The measure of a point p is equal to the measure of the starting point p_i of segment s_i on which the point p is located, plus the Euclidean distance $d(p_i, p)$.The representation of a moving object on a polyline is a four-tuple $mop = (pl, m, plspd, t)$. Here, $pl \in R$ is the polyline on which the moving object is located; m is the measure giving the moving object's location on *pl; plspd* is the moving object's signed speed along the polyline; and t is the time when the preceding values are valid.

3.2 Index Structure

PPFN*-tree consists of static part and dynamic part. The former contains a 2D R*-tree built on road sectors for managing the road networks and a hash structure H_1. The latter contains a set a TB*-trees indexing objects' movement history trajectory along the polylines, a set of basic HTPR*-tree indexing the position of moving objects after recent update, and a hash table H_2. In order to improve update and query performance, we use H_1 in static part containing entries of the form <*polyid, tree1pt, tree2pt*>, where *polyid* is the polyline identification, *tree1pt* is a pointer to the corresponding TB*-tree, and *tree2pt* is a pointer to the corresponding basic HTPR*-tree.

Figure 2 illustrates the overall data structures of PPFN*-tree. Note R_{road} is a 2D R*-tree for managing the road networks. Each leaf node entry of R_{road} points to a TB*-tree and a basic HTPR*-tree. Note R_{tbi} ($1 \le i \le n$) is a TB*-tree for managing history trajectories of moving objects from some past time until the time of the most recent position sample of each object $o(t^o_{mru})$ moved on polyline S_i ($1 \le i \le n$). R_{hi}($1 \le i \le n$) is the basic HTPR*-tree which indexes position information since the most recent update instant of each object (t^o_{mru}) moving on polyline S_i($1 \le i \le n$).

In $R_{road,}$ leaf nodes contain the information <*mbb, polypt*, *tree3pt, tree4pt*> where *mbb* is the MBB of the polyline, *polypt* points to the real representation of the polyline, *tree3pt* points to TB*-tree of object moved on that polyline, and *tree4pt* points to basic HTPR*-tree of object moving on that polyline. Internal nodes have the following information <*mbb*, *childpt*>, where *mbb* is the MBB that contains all MBBs of the entries in the child node, and *childpt* is a pointer to the child node.

Note R_{tbi} ($1 \le i \le n$) is a TB*-tree for managing history trajectories of moving objects from some past time until the time of the most recent position sample of each object $o(t^o_{mru})$ moved on polyline S_i($1 \le i \le n$). The node structure is as follows:

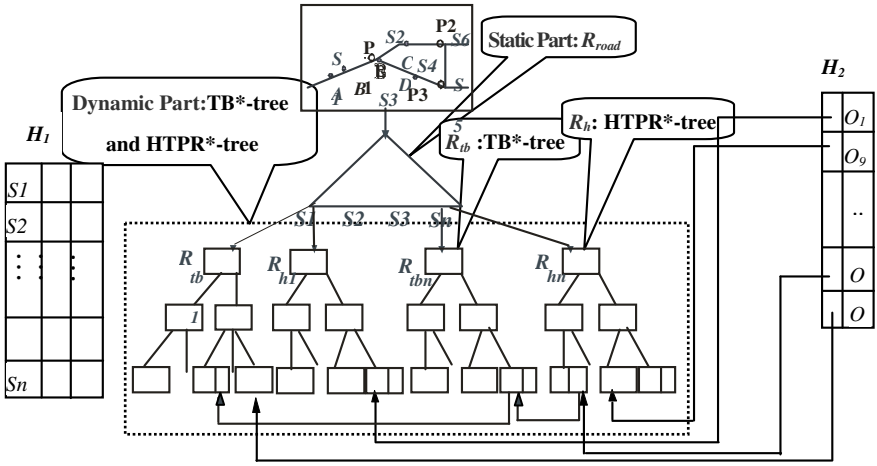


Fig. 2. Overall data structure of PPFN*-tree

$N^{nonleaf}$= [ptr_{parent}, (ptr_1, MBB_1), (ptr_2, MBB_2), …, (ptr_k, MBB_k)]
N^{leaf}= [*Oid*, *polyid*, ptr_{parent}, ptr_{prev}, ptr_{next}, (m_1, t_1), …, (m_k, t_k)]

Where ptr_i ($1 \le i \le k$) points its childe node with MBB_i, and ptr_{parent} points its parent node. Ptr_{parent} is used for improvement of insertion performance by bottom-up update when new data is inserted into index.

In the leaf node, its entry contains not *MBB* but location of moving object. *Oid i*s the identifier of the moving object, and *polyid* is the identifier of the polyline where the object is located. The measure (*m*) of a point *p* located on a polyline is the distance which measured along the polyline from the start point p_0 to point *p*. (m_i, t_i) implies the measure of the objects inside the given polyline are m_i at time t_i. The leaf nodes of TB*-tree are linked with other leaf nodes belonging to the same object. Ptr_{prev} and ptr_{next} are used for managing linked list. Ptr_{prev} is a pointer that points to the last node either in the same R_{tb} or in the different R_{tb}, and ptr_{next} points to the next node either in the same $R_{tb,}$ or in the different R_{tb}, or in the R_h. This information makes it easy to access whole trajectory with minimal cost by visiting an arbitrary leaf node.

R_{hi} is a basic HTPR*-tree to manage position information since the most recent update instant of each object (t^o_{mru}) moving on polyline S_i($1 \le i \le n$). In order to support querying from the past to the future, R_{hi} should be associated with R_{tbj}($1 \le j \le n$) through pointer. We add *ptr* to leaf node entry of R_{hi} which points to a leaf node of R_{tbj} including the most recent history segment. So, the structure of each leaf node entry of R_{hi} is of the form *(oid, tpp, ptr, t*). Here, *tpp*=(*m, v*), with the *m* and *v* being the measure and velocity, respectively, of the object located on polyline S_i at creation or update time *t.* The structure of each non-leaf node entry is in the form of *(ptr, tpbr, st1, st2).* Here *ptr* is a pointer that points to the child node. *St1* is the minimal creation or update time of moving objects included in the child node pointed by *ptr,* and *st2* is the maximum value

compare with *st1*. The *tpbrs* of the Basic HTPR*-tree are bound time-parameterized points which bound objects since time *st1*.

The item in hash table H_2 is defined as vector <*oid, ptr*>, where *oid* denotes the identifier of moving object, and *ptr* denotes physical offset of the leaf node in R_{hi} which object entry locates.

3.3 Insertion

Inserting a new moving object into PPFN*-tree involved R_h, hash table H_1 and H_2. It is carried out by three steps: (1) Based on the polyline identification *polyid* where the object *o* is located, we can get R_{hi} where the object *o* should be inserted through hash structure H_1; (2) A entry (*oid, tpp, ptr, st*) (*ptr* is null) describing the object *o* is inserted in R_{hi} and get the leaf node where entry (*oid, tpp, ptr, st*) is located. (3) The object *o* is registered with its *oid* in H_2 and linked to the leaf node of R_{hi} storing the object *o* with pointer. The detailed algorithm is given in algorithm 1.

Algorithm 1. Insert (*R, o, polyid*)

/*Input: *o* is a moving object with *oid, m, v* and *t*; *R* is the PPFN*-tree, *polyid* is the polyline identification that *o* located*/

1. get the R_{hi} that *o* should be inserted
2. invoke Insert (R_{hi}, *o*)
3. achive the leaf node of *o* stored in R_{hi}
4. register *o* with its *oid* in H_2, link H_2 with leaf node of R_{hi} using *ptr*/*

ENDInsert

3.4 Update

When a moving object *o* with its object identifier reports a new position and velocity to the system, update may be caused by the following three kinds of situation. Firstly, the velocity is changed. Secondly, the allowed position precision threshold is exceeded. Lastly, the new position reported by the object *o* is outside the polyline S_i but inside another polyline S_j, that is the polyline identification *polyid* is changed. In the first two kinds of situation, *o* and *o' (after changed of o)* are located in the same polyline. So, we can get R_{hi} where *o'* will be insert and R_{tbi} where the history segment of *o* will be inserted through H_2, update of PPFN*-tree need only update R_{hi} and insert a history segment to R_{tbi}. Of course, the leaf node entry of the object *o* in R_{hi} can be obtained through H_2 and updated in a bottom-up manner strategy.

Algorithm 2 describes update procedure of PPFN*-tree indexing objects that are moving on the same polyline.

Algorithm 2. Update (*R , o, o', polyid*)

/*Input: *o* is a moving object with *oid, m, v* and *st*; at time *st',* it is changed to *o'; R* is the PPFN*-tree, *polyid* is the polyline identification that *o* located */

1. get the R_{tbi} where history trajectory segment *of o* should be inserted through $R.H_1$
2. get leaf node entry *e1* of *o* in R_{hi} through $R.H_2$
3. invoke R_{hi}.update(*o, o'*)
4. get leaf node entry *e2* of *o'* in R_{hi}
5. get history trajectory segment *e3=(st, st', m, m')*
6. if object *o* has no history trajectories in R_{tbi}
7. insert new segment e3 in R_{tbi} with top-down
8. create pointer of leaf node where *o'* is located in R_{hi} with leaf node where *e3* is located in R_{tbi}
9. else
10. get leaf node *n1*where the last recent history trajectory segment of *o* is located in R_{tbi} through *e1* of *o* in R_{hi}
11. if *n1* has space
12. Insert new segment *e3* in *n1*
13. else
14. create new leaf node *n2* for new segment *e3*
15. find a non-full parent node *n3* of *n1*
16. insert *n2* in tree rooted by *n3* through right-most path
17. create ptr_{prev} pointers of *n2* to *n1* in R_{tbi}, and ptr_{next} pointers of *n2* to entry *e2* of *o'* in R_{hi}

ENDUpdate

3.5 Search Procedure

3.5.1 Spatio-Temporal Range Query

PPFN*-tree can support spatio-temporal range query (given a query window $w=(x_1, y_1, x_2, y_2, t_1, t_2)$, the query is "find all objects that have lain within the area $r=(x_1, y_1, x_2, y_2)$ during the time interval $t=(T_1, T_2)$").

In the range query, when the temporal dimension is zero extent, a special case of range query so-called time-slice query is shown in Figure 3. Figure 4 describes timeslice query and spatio-temporal range query of objects moving on one of polylines on road network.

For the range query, the algorithm receives a spatio-temporal query window *w* and performs in the following three steps. In the first step, a search in the top R*-Tree is performed to find the polylines' MBBs that intersect the spatial query window *r*. Then, the intervals where the polyline intersects *r* are searched using the real polyline

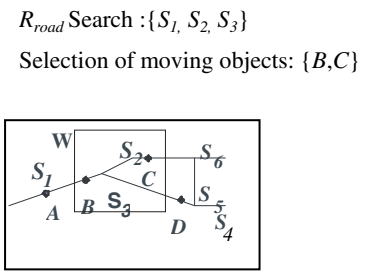


Fig. 3. Time–slice Query of PPFN*-tree

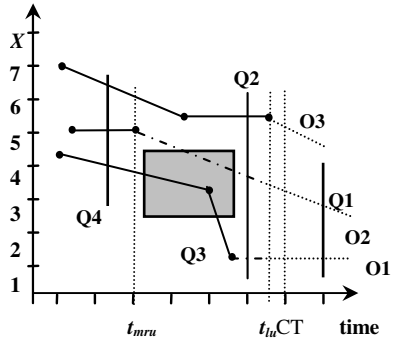


Fig. 4. Querying the Positions of Moving Objects

representation. The result is a set of windows w'= {($M11$, $M12$, T_1, T_2), . . ., ($Mn1$, $Mn2$, T_1, T_2)}, where n is the set size, n ≥ 1, and the interval (T_1, T_2) is the query time interval t. Here, the windows are disjoint and ordered. Finally, given this set of windows w', each w' is performed query in corresponding R_{tbi} or R_{hi} pointed by polyline S_i.

Spatio-temporal range query in PPFN*-tree is bound up with the query time interval t=(T_1, T_2). The detailed search procedure is as follows:

Algorithm 3. RQuery(R, r, T_1, T_2)

/*Input: R is the PPFN*-tree, r is the query spatial area, (T_1, T_2) is the query time interval*/

1. get $R.R_{road}$
2. performing a search in $R.R_{road}$ to find the polylines' MBBs that intersect r
3. for each S_i acquired in step 2
4. change search w to a set of w'= {($M11$, $M12$, T_1, T_2),…, ($Mn1$, $Mn2$, T_1, T_2)} in corresponding polyline S_i
5. if $T2$< $R_{tbi.}$*root.st1*,
6. for each w'=($Mj1$, $Mj2$ T_1, T_2) invoke $R_{tb.}$RQuery$_{.}$($R_{tbi,}$ $Mj1$, $Mj2$, T_1, T_2)
7. else if $T1$>$R_{hi.}$*root. st2*
8. for each ($Mj1$, $Mj2$ T_1, T_2) invoke $R_{h.}$RQuery$_{..}$($R_{hi,}$ $Mj1$, $Mj2$, T_1, T_2)
9. else range query is implemented in both R_{tbi} and R_{hi}

End RQuery(R, r, $t1$, $t2$)

3.5.2 Trajectory Query

Trajectory query is to extract information related to moving objects' trajectories, e.g., "What were the trajectories of trains after they left Wuhan between 5 and 12 today, in the next hour?" We have to (a) select the objects, and (b) select the partial trajectory of each obtained object. In PPFN*-tree, selection of objects can occur by range query and topological query, or obtained by objects identifiers directly. So trajectory query in

PPFN*-tree can be realized by selecting object records in H_2 and getting pointer to extract objects' trajectories during $t=(T_1, T_2)$ in the corresponding R_h and R_{tb}.

3.5.3 Topological Query

Query of the form "find all objects that *enter, leave, cross,* or *bypass* a given area $r=(x_1, y_1, x_2, y_2)$, during the time interval $t=(t_1, t_2)$" is called *topological query* [3].

The *topological query* algorithm consists of three steps: (1) We can obtain polylines intersecting with area r via R_{road}, and area $r=(x_1, y_1, x_2, y_2)$ is changed into $r' = (M_1, M_2)$ in given R_s; (2) Each leaf node entry that (T_1, T_2) intersects with t in every R_s obtained in the first step is examined, and we can obtain the object's measure in t_1 and t_2; (3) The result in step 2 can be used to estimate *topological query*. Figure 4 shows object O1 *leaves* Q3, but O2 *enters* Q3.

4 Performance Study

4.1 Experimental Setting and Details

In this section, we evaluate the performance of the PPFN*-tree with the MON-tree, the PPFI and the R^{PPF}-tree. In all our experiments, we used the network-based moving objects generator proposed in [24]. The generator takes a map of a real road network as input (our experiment is based on the map of Oldenburg including 7035 segments). The positions of the objects are given in two dimensional X-Y coordinates. Since our experiment use edge (polyline) oriented model, we transform X-Y coordinates to the form of (*ployid*; *pos*), where *ployid* denotes the polyline identifier and *pos* denotes the object relative position on the polyline. After that, the total number of polylines is 3803.

4.2 Performance Analysis

- **Update Cost Comparison**

Figure 5 compares the average update cost of the PPFN*-tree, the PPFI and the R^{PPF}-tree as a function of the number of updates. The update cost of the PPFN*-tree increases as the num of updates, but it does not as much as that of the PPFI and the R^{PPF}-tree. This is due to the fact that the HTPR*-tree and TB*-tree of PPFN*-tree adopt bottom-up update strategy to avoid the excessive node accesses, but the PPFI and the R^{PPF}-tree adopt top-down update strategy. Moreover, a history trajectory after the most recent update instant in the R^{PPF}-tree is stored in several entries and even in several nodes, which should be modified when an update occurs. This greatly enhances the cost of update operation of the R^{PPF}-tree. At the same time, a new trajectory inserted into the R^{PPF}-tree also causes *time split* of node, which also reduce the update performance.

- **Query Cost Comparison**

In order to study the deterioration of the indices with time, we measure the performance of the PPFN*-tree, MON-tree and the PPFI using the same query workload.

First, we compare trajectory query performance of the PPFN*-tree with that of PPFI. Figure 6 shows trajectory query performance. We find the PPFN*-tree is more efficient than PPFI under trajectory query, since moving object trajectories were stored in leaf nodes of TB*-tree and basic HTPR*-tree which can be obtained through pointer in H_2 directly. Because the TB*-tree *strictly preserves trajectories* such that a leaf node only contains segments belonging to the same trajectory, trajectory query of the PPFN*-tree will access less leaf nodes than that of the PPFI.

Then, we used range query which tries to find all objects that are moving during a given time interval in a given area. We are interested in the behavior of the indexes according to the follow variables:

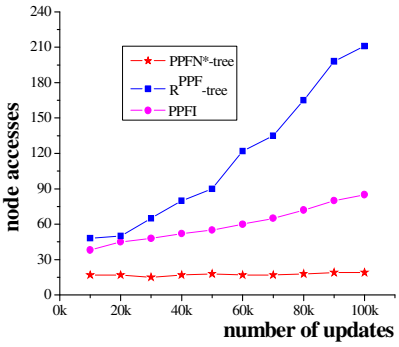


Fig. 5. Update Cost Comparison

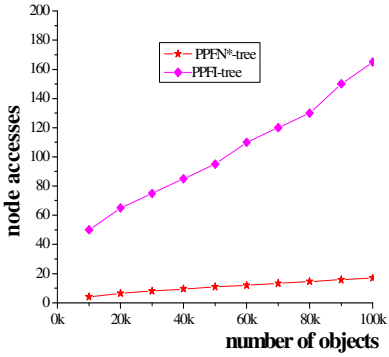
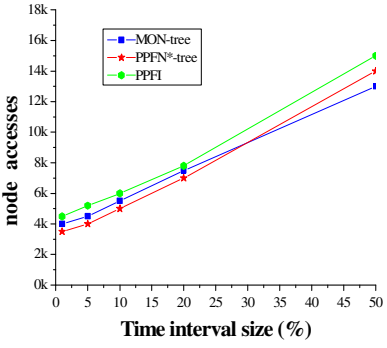


Fig. 6. Trajectory Query Cost Comparison



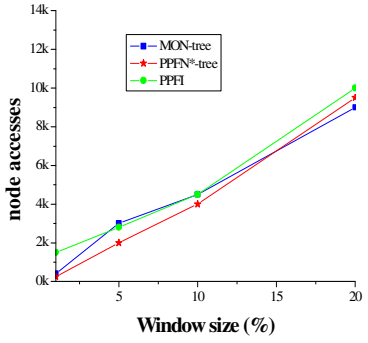


Fig. 7. Range Query Cost Comparison

- Size of the query time interval. We generated queries with a range of 1%, 5%, 10%, 50%, and 100% of the total data set time interval.
- Size of the query window. We generated queries with a range of 1%, 5%, 10%, and 20% of the total data set space.

For each combination of the two variables, we generated randomly 100 queries. The performance of the indexes is then compared by the average number of disk accesses for executing the queries. Figure 7 shows the influence of the query time interval size and the query window size in the performance of queries. As can be seen from these figures, the PPFN*-tree has a linear behavior with respect to the increase of these two variables. For a small range of query interval, the PPFN*-tree shows superior query performance over the MON-tree and the PPFI. However, with the query window size increasing in the largest query time interval size, the query performance deterioration of the PPFN*-tree is more than that of the MON-tree. Because range query of the bottom 2D R-Trees in the MON-tree is superior to that of TB*-tree in the PPFN*-tree. Similarly, with the query time interval size increasing in the largest query window size, the query performance deterioration of the PPFN*-tree is more than that of the MON-tree.

5 Conclusion

In this paper, we develop a novel index structure named PPFN*-tree which consists of a 2D R*-tree which is built on polylines describing road sectors for managing the road networks, a set of TB*-trees indexing objects' movement history trajectory along the polylines, and a set of basic HTPR*-tree indexing the position of moving objects after recent update. PPFN*-tree can not only support past trajectory query and present position query, but also support future predictive query. Extensive experiments prove that the update performance of PPFN*-tree is better than those of the PPFI and the R^{PPF}-tree, trajectory query performance of the PPFN*-tree are significantly improved compared with those of the PPFI. At the same time, for a small range of query interval, the PPFN*-tree shows superior query performance over the MON-tree and the PPFI.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (Grant No.90718027 and Grant No.41171315).

References

[1] Pfoser, D.: Indexing the Trajectories of Moving Objects. IEEE Data Engineering Bulletin 25(2), 2–9 (2002)
[2] Kollios, G., Gunopulos, D., Tsotras, V.J.: On indexing mobile objects. In: Proc. of ACM Symp. on Principles of Database Systems (PODS), pp. 261–272 (1999)
[3] Jensen, C.S., Pfoser, D.: Indexing of network constrained moving objects. In: Proc. of the 11th Intl. Symp. on Advances in Geographic Information Systems (2003)
[4] Frentzos, E.: Indexing objects moving on fixed networks. In: Hadzilacos, T., Manolopoulos, Y., Roddick, J., Theodoridis, Y. (eds.) SSTD 2003. LNCS, vol. 2750, pp. 289–305. Springer, Heidelberg (2003)

[5] Victor, T.D.A., Ralf, H.G.: Indexing the Trajectories of Moving Objects in Networks. GeoInformatica 9(1), 33–60 (2005)
[6] Kim, K.-S., Kim, S.-W., Kim, T.-W.: Fast indexing and updating method for moving objects on road networks. In: Proc. of the 4th Intl. Conf. on Web Information Systems Engineering Workshops, pp. 34–42 (2003)
[7] Fang, Y., Cao, J.: Indexing the Past, Present and Future Positions of Moving Objects on Fixed Networks. In: Intl. Conf on Computer Science and Software Engineering, pp. 524–527 (2008)
[8] Pfoser, D., Jensen, C.S., Theodoridis, Y.: Novel Approaches to the Indexing of Moving Object Trajectories. In: Proc. of the 26th Intl. Conf. on Very Large Databases, pp. 395–406 (2000)
[9] Tao, Y., Papadias, D.: MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In: Proceedings of the International Conference on Very Large Databases, VLDB (2001)
[10] Tayeb, J., Ulusoy, O., Wolfson, O.: A Quadtree-Based Dynamic Attribute Indexing Method. The Computer Journal 41(3), 185–200 (1998)
[11] Kollios, G., Gunopulos, D., Tsotras, V.J.: On Indexing Mobile Objects. In: ACM PODS, pp. 261–272 (1999)
[12] Papadopoulos, D., Kollios, G., Gunopulos, D., Tsotras, V.J.: Indexing Mobile Objects on the Plane. In: MDDS, pp. 693–697 (2002)
[13] Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the Positions of Continuously Moving Objects. In: ACM SIGMOD, pp. 331–342 (2000)
[14] 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, pp. 790–801 (2003)
[15] Jensen, C.S., Lin, D., Ooi, B.C.: Query and Update Efficient B+-Tree Based Indexing of Moving Objects. In: VLDB, pp. 768–779 (2004)
[16] Chen, S., Ooi, B.C., Tan, K.L., Nacismento, M.: ST2B-tree: A Self-Tunable Spatio-Temporal B+-tree Index for Moving Objects. In: ACM SIGMOD, pp. 29–42 (2008)
[17] Fang, Y., Cao, J., Wang, J., Peng, Y., Song, W.: HTPR*-Tree: An Efficient Index for Moving Objects to Support Predictive Query and Partial History Query. In: Wang, L., Jiang, J., Lu, J., Hong, L., Liu, B. (eds.) WAIM 2011. LNCS, vol. 7142, pp. 26–39. Springer, Heidelberg (2012)
[18] Fang, Y., Cao, J., Peng, Y., Chen, N.: Indexing Partial History Trajectory and Future Position of Moving Objects Using HTPR*-Tree. In: Yu, H., Yu, G., Hsu, W., Moon, Y.-S., Unland, R., Yoo, J. (eds.) DASFAA Workshops 2012. LNCS, vol. 7240, pp. 229–242. Springer, Heidelberg (2012)
[19] Sun, J., Papadias, D., Tao, Y., Liu, B.: Querying about the past, the present and the future in spatio-temporal databases. In: ICDE, pp. 202–213 (2004)
[20] Lin, D., Jensen, C.S., Ooi, B.C., Saltenis, S.: Efficient indexing of the historical, present, and future positions of moving objects. In: MDM, pp: 59–66 (2005)
[21] Pelanis, M., Saltenis, S., Jensen, C.S.: Indexing the Past, Present and Anticipated Future Positions of Moving Objects. ACM TODS 31(1), 255–298 (2006)
[22] Fang, Y., Cao, J., Zeng, C., Chen, N.: Indexing the Past, Present and Future Positions of Moving Objects Using PPFI*. In: Proc. of the 8th Intl. Conf. on Networked Computing and Advanced Information Management, pp. 314–320 (2012)
[23] http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/