Indexing the Past, Present and Future Positions of Moving Objects on Fixed Networks

Ying Fang\textsuperscript{1}, Jiaheng Cao\textsuperscript{1},
\textsuperscript{1}School of Computer, Wuhan University
Wuhan 430072, P. R. China
fy\_fy003@sina.com  jhcao@whu.edu.cn

Yuwei Peng\textsuperscript{1}, Liwei Wang\textsuperscript{2}
\textsuperscript{2}International School of Software, Wuhan University
Wuhan 430072, P. R. China
sandsman@126.com  davilisa@yahoo.com.cn

Abstract

The development of a spatio-temporal access method suitable for objects moving on fixed networks is a very attractive challenge due to a large number of applications dealing with this type of objects. In this paper, we propose a novel indexing technique named PPFI which consists of a 2D R*-tree, a forest of 1D R*-tree, and a hash structure. PPFI not only supports queries related to the past positions or trajectories of moving objects on fixed networks, but also provides an efficient update mechanism, stores current positions and supports predictive query. The performance study, comparing this access method with FNR-Tree, STR-Tree under range query and trajectory query, shows that PPFI outperforms them.

1. Introduction

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

According to reference [1], applications dealing with moving objects can be grouped into three movement scenarios, namely unconstrained movement, constrained movement, and movement in transportation networks. Most works for indexing moving objects assume free movement of the objects in space. But in many applications, movement occurs on fixed road network.

In this paper, we address the problem of indexing moving objects on fixed network. We propose a novel indexing method named Past-Present-Future-Index (PPFI) to store current positions, past trajectories, and predict near future positions of moving objects on fixed networks. PPFI 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. Compared to the existing index structures that can only index the past trajectory or the current and future position of moving objects, PPFI has much stronger functions and practical applications.

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 PPFI, shows the index structure of PPFI and the corresponding algorithms. Section 4 reports on the performance evaluation. The conclusion is given in Section 5.

2. Related works and motivations

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

Some index structures have been proposed for moving objects. They are classified into two categories: one of them is to handle past positions or trajectories of moving objects [2][3], and the other is to handle current and future positions [4][5]. Each of them requires quite different functions and approaches.
In order to manage moving objects on fixed network, some special index structures have been developed. For example, C. S. Jensen et al. [6] 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 [7] 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 [8] 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 and cannot support frequent index update. Therefore, query related to the current positions and near future positions of moving objects could not be realized in these index structures.

IMORS [9] 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.

Our work aims to provide an indexing structure to support querying the past, present and future positions of moving objects on fixed networks. This indexing structure should describe the movement of moving objects and support frequent update of moving objects positions.

3. PPFI

3.1. Data modeling and update policy

As known, moving objects frequently change their positions. If the objects submit update requests to moving object database system at every time unit, the storage costs and communication costs are very expensive. An appropriate data model and update policy is thus very important. In this work, we use a linear prediction method to model the objects’ movement and a linear update policy to record the objects’ movement.

3.2. Index structure

PPFI consists of static part and dynamic part. The former contains only a 2D R*-tree built on road sectors for managing the fixed networks. The latter contains a set of 1D R*-tree indexing the time interval for the past trajectory of moving object, and a hash structure describing the recent state of moving objects and predicting near future positions of moving objects.

Figure 1 illustrates the overall data structures of PPFI. Note \( R_{road} \) is a 2D R*-tree for managing the fixed networks. Each leaf node entry of \( R_{road} \) points to a 1D R*-tree, \( R_e \). \( H_{data} \) stores the most recent update time \( T \), the recent update position \( M \) and velocity \( V \), and the identifier of the polyline where moving object located currently.

![Figure 1 Overall data structure of PPFI.](image)

Each \( R_e \) in Figure 1 indexes the movement of the objects inside a polyline. The structure of each leaf node entry of \( R_e \) is of the form \((Oid, Plid, T_1, T_2, M_1, M_2, P_1, P_2)\), and the structure of each non-leaf node entry of \( R_e \) is of the form \((Pointer to child node, T_1, T_2)\). Here \( Oid \) is the identifier of the moving object, and \( Plid \) is the identifier of the polyline where the object is located. The movement described by the measure interval \((M_1, M_2)\) and a time interval \((T_1, T_2)\) implies the measure of the objects inside the given polyline are \( M_1 \) at time \( T_1 \) and \( M_2 \) at time \( T_2 \). The measure \( M \) of a point \( p \) located on a polyline is the distance which measured along the polyline from the start point to point \( p \). \( P_d \) is a pointer that points to the object record in \( H_{data} \)(the entry is just inserted, otherwise points to subsequence entry of the same object), and \( P_l \) points to the last entry either in the same \( R_e \) or in the different \( R_e \).

3.3. Insertion algorithm

Inserting a new moving object into PPFI is carried out by three steps: (1) A new object registers its oid in \( H_{data} \); (2) Based on the object coordinates, we can get the identifier of the polyline where the object is located and the corresponding measure \((m)\). At the same time, the current time and the object’s current velocity are also recorded; (3) A new entry is inserted in leaf node of the \( R_e \) that pointed by pointer. Here \( M_1 \) and \( T_1 \) of the new entry are the
measure just reported and the current time instant, respectively. $T_2$ is $T_{\text{move}}$, and $M_2$ is obtained by formal $M_1^+(T_{\text{move}}-T_i)*V$. $P_i$ is null, $P_d$ points to the new record in $H_{\text{data}}$ and $\text{Pointer}$ of the new record in $H_{\text{data}}$ points to the new entry in $R_c$.

3.4. Update algorithm

When a moving object with its object identifier reports a new position and velocity to the system, we should retrieve the record from $H_{\text{data}}$ by the $\text{oid}$. An update is issued if the velocity is changed or the allowed position precision threshold is exceeded.

Since this record has the $\text{pointer}$ to the leaf node entry of the corresponding $R_c$, we can directly access the leaf node entry and modify it. Sometimes a new leaf node entry needs to be created in the same $R_c$ or the different $R_c$. The detailed process is as follows: (1) When an update is issued and the object still on the same polyline, a new leaf node entry needs to be created in the same $R_c$. $M_1$ of the new entry is the measure just reported, and $T_1$ is the current update instant. $T_1$ is $T_{\text{move}}$, and $M_1$ is obtained by formal $M_1^+(T_{\text{move}}-T_i)*V$. $M_2$ and $T_2$ of the last entry are the same as $M_1$ and $T_1$ of the new entry. At the same time, pointers among the new entry, the last entry and $H_{\text{data}}$ are created.

(2) If an update is issued and the new position is not on the same polyline, the update process is a little complex compared with the above case. Firstly, we should obtain the polyline, where the new location is placed, and the measure. Secondly, two entries (e and $e'$) are inserted in $R_c$ pointed by the new polyline. Entry $e$ is obtained by the same way as the new entry in the first case. Entry $e'$ records the trajectory from the start point (the object moves with positive velocity, otherwise the end point) of the new polyline to the position just reported. Thirdly, $M_2$ of the last entry in old $R_c$ is changed into the maximal measure (the velocity is positive, otherwise zero) of old polyline, and $T_2$ the corresponding time. And finally, pointers among the new entry, the last entry and $H_{\text{data}}$ are created.

3.5. Search procedure

3.5.1. Spatio-temporal range query. PPFI can support spatio-temporal range query (given a query window $w=(x_l, y_l, x_u, y_u)$, the query is “find all objects that have lain within the area $r=(x_l, y_l, x_u, y_u)$ 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 named time-slice query is performed. First, we can search the polyline intersecting or contained within area $r$ via $R_{\text{move}}$. Secondly, intervals where the polyline intersects $r$ are searched using the real polyline representation, and query $w$ is changed into $w=(m_1, m_2, t_1, t_2)$ in obtained $R_c$. Finally, each leaf node entry that $(T_1, T_2)$ contains $t_i$ in all obtained $R_c$ is examined if the object’s measure is contained within $(m_1, m_2)$. Figure 2 describes range query, for example, query $Q_1$ gets object O1 and O2.

![Figure 2 Spatio-temporal Range Query Example of PPFI](image)

3.5.2. Trajectory query. Trajectory query is to extract information related to past trajectories, e.g., “What were the trajectories of trains after they left Hubei between 6 and 11 today, in the next hour?” We have to (a) select the objects, and (b) select the partial trajectory of each obtained object.

In PPFI, selection of objects can occur by range query and topological query, or obtained by objects identifiers directly. So trajectory query in PPFI can be realized by selecting object records in $H_{\text{data}}$ and getting $\text{Pointer}$ to extract objects’ trajectories during $t=(t_1, t_2)$ in the corresponding $R_c$.

3.5.3. Topological query. Query of the form “find all objects that enter, leave, cross, or bypass a given area $r=(x_l, y_l, x_u, y_u)$, 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 $R_{\text{move}}$ and area $r=(x_l, y_l, x_u, y_u)$ is changed into $r=(m_1, m_2)$ in given $R_c$; (2) Each leaf node entry that $(T_1, T_2)$ intersects with $t$ in every $R_c$ obtained in the first step is examined, and we can obtain the object’s measure in $t$ and $t_2$; (3) The result in step 2 can be used to estimate topological query. Figure 2 shows object O1 enters Q2, but O2 leaves Q2.
3.5.4. Predictive query. PPFI supports predictive query using a linear update policy. Because $H_{data}$ stores the most recent update time $T_1$, update position $M_1$, velocity $V$, and the identifier of polyline $(Plid)$ where moving object located currently, the measure $M_2$ in the future time instant $T_2$ can be obtained by formal $M_2=M_1+ (T_2-T_1)*V$ if the object does not leave the current polyline.

4. Performance study

We have done an experimental evaluation in order to examine the performance of our index structure. The performance study was based on synthetic datasets created using a network-based data generator [10] and the railroads (rrline) networks [11]. Our results are compared with those of FNR-Tree and STR-Tree [3].

In our experiment, the evaluation showed in Figure 3 (A) and (B) implies that PPFI supports range query much more efficiently than FNR-Tree. Because the size of PPFI is much smaller than that of FNR-Tree, the time needed for executing queries in PPFI is reduced greatly, and query efficiency are improved. Figure 3 (C) shows trajectory query performance of PPFI compared with STR-Tree. We find PPFI is more efficient than STR-Tree under trajectory query, since moving object trajectory can be obtained through pointer in $H_{data}$ of PPFI directly.

5. Conclusions

We proposed a new index structure (PPFI) for moving objects on networks. PPFI stores the complete trajectories of the objects moving in networks. It not only supports range query, topological query and trajectory query related to the past and present positions of moving objects, but also predicts future positions of moving objects. We have experimentally evaluated the proposed index structure with generated data sets. It is found that PPFI have better scalability and performance than FNR-Tree and STR-Tree.

References