

# An Efficient Trajectory Index Structure for Moving Objects in Location-Based Services\*

Jae-Woo Chang<sup>1</sup>, Jung-Ho Um<sup>1</sup>, and Wang-Chien Lee<sup>2</sup>

<sup>1</sup> Dept. of Computer Eng., Chonbuk National Univ.,  
Chonju, Chonbuk 561-756, Korea

{jwchang, jhum}@dblab.chonbuk.ac.kr

<sup>2</sup> Dept. of CS&E., Pennsylvania State Univ.,  
University Park, PA 16802

**Abstract.** Because moving objects usually moves on spatial networks, efficient trajectory index structures are required to gain good retrieval performance on their trajectories. However, there has been little research on trajectory index structure for spatial networks, like road networks. In this paper, we propose an efficient trajectory index structure for moving objects in Location-based Services (LBS). For this, we design our access scheme for efficiently dealing with the trajectories of moving objects on road networks. In addition, we provide both an insertion algorithm to store the initial information of moving object trajectories and one to store their segment information. We also provide a retrieval algorithm to find a set of moving objects whose trajectories match the segments of a query trajectory. Finally, we show that our trajectory access scheme achieves about one order of magnitude better retrieval performance than TB-tree.

## 1 Introduction

In general, spatial databases has been well studied in the last two decades, resulting in the development of numerous spatial data models, query processing techniques, and index structures for spatial data [Se99]. Most of the existing work considers Euclidean spaces, where the distance between two objects is determined by the ideal shortest path connecting them in the spaces. However, in practice, objects can usually move on road networks, where the network distance is determined by the length of the real shortest path connecting two objects on the network. For example, a gas station nearest to a given point in an Euclidean space may be more distant in a road network than another gas station. Therefore, the network distance is an important measure in spatial network databases (SNDB). Recently, there have been some studies on SNDB for emerging applications such as location-based service (LBS) [B02, HKT02, F03, PZM03, SJK03, SKS03]. First, Speicys et al. [SJK03] dealt with a computational data model for spatial network. Secondly, Shahabi et al. [SKS03] presented k-nearest neighbors (k-NN) query processing algorithms for SNDB. Finally, Papadias et al. [PZM03] designed a novel index structure for supporting query processing algorithms.

---

\* This work was supported by the Korea Research Foundation Grant. (KRF-2003-013-D00094).

Because moving objects usually moves on spatial networks, instead of on Euclidean spaces, efficient index structures are required to gain good retrieval performance on their trajectories. However, there has been little research on trajectory access schemes for spatial networks, like road networks. In this paper, we propose an efficient trajectory index structure for moving objects in Location-based Services (LBS). For this, we design our access scheme for efficiently dealing with the trajectories of moving objects on road networks. In addition, we provide both an insertion algorithm to store the initial information of moving object trajectories and one to store their segment information. We also provide a retrieval algorithm to find a set of moving objects whose trajectories match the segments of a query trajectory.

The rest of the paper is organized as follows. In Section 2, we discuss background and motivation for our work. In Section 3, we propose a trajectory access scheme for moving objects. In Section 4, we provide the performance analysis of our access scheme. Finally, we draw our conclusion in Section 5.

## 2 Background and Motivation

To our knowledge, there has been little research on trajectory access schemes for spatial networks. So we overview both a major index structure for spatial networks and a predominant trajectory index structure for Euclidean spaces. First, Papadias et al. [PZM03] proposed a network storage scheme for spatial network databases (SNDB) by separating the network itself from entity datasets. They employ a disk-based network representation that preserves connectivity and location, while spatial entities are indexed by respective spatial access methods for supporting Euclidean queries and dynamic updates. Their network storage scheme consists of three components, i.e., adjacency component, poly-line component, and network R-tree. The adjacency component captures the network connectivity by which the adjacency lists of nodes being close in space according to their Hilbert values are placed in the same disk. The poly-line component stores the detailed poly-line representation of each segment in the network. The last network R-tree component indexes the poly-lines' MBRs (Minimum Bounding Rectangles) and supports queries exploiting the spatial properties of the network. In addition, Pfooser et al. [PJT00] proposed a hybrid index structure which preserves trajectories as well as allows for R-tree typical range search in Euclidean spaces, called TB-tree (Trajectory-Bundle tree). The TB-tree has fast accesses to the trajectory information of moving objects, but it has a couple of problems in SNDB. First, because moving objects move on a predefined spatial network in SNDB, the paths of moving objects are overlapped due to frequently used segments, like downtown streets. This leads to a large volume of overlap among the MBRs of internal nodes. Thus, the TB-tree no longer achieves good performance on retrieving moving object trajectories in SNDB. Secondly, when a moving object moves on the predefined spatial network for a long time, the dead space for the moving object trajectory is highly increased because the TB-tree constructs a three-dimensional MBR including time. This leads to a large volume of overlap with other objects' trajectories. Thus, the TB-tree results in poor retrieval performance on moving object trajectories in SNDB, due to its small discrimination capability.

### 3 Trajectory Access Scheme for Moving Objects

#### 3.1 Architecture of Trajectory Access Scheme

Because moving objects change their locations continuously on road networks, the amount of trajectory information for a moving object is generally very large. To solve the problems of TB-tree as mentioned in section 2, we propose a new signature-based access scheme which can have fast accesses to moving object trajectories. Figure 1 shows the structure of our signature-based trajectory access scheme.

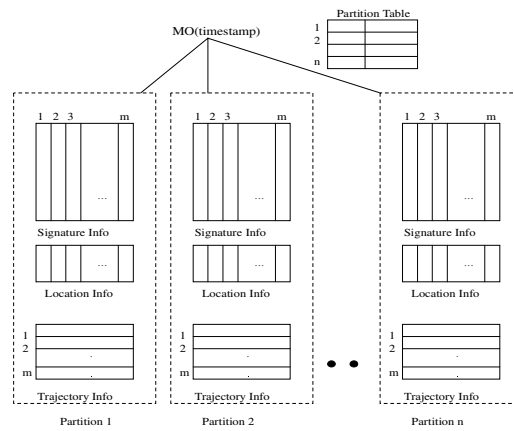


Fig. 1. Architecture of trajectory access scheme

The main idea of our trajectory access scheme is to create and maintain partitions which store the predefined number of moving object trajectories and their signatures together in the order of their start time. There are a couple of reasons for using partitions. First, because a partition includes the fixed number of moving object trajectories, it is possible to construct a bit-sliced signature file which enables good retrieval performance. Secondly, because a partition can be accessed independently to answer a trajectory-based query, it is possible to achieve better retrieval performance by searching partitions in parallel. Finally, because a partition is created and maintained depending on its start time, it is possible to efficiently retrieve the trajectories of past moving objects on a given time. Therefore, our trajectory access scheme has three advantages. First, our access scheme is not affected by the overlap of moving objects' paths and never causes the dead space problem because it is not a tree-based structure like TB-tree. Secondly, our access scheme well supports a complex query containing a partial trajectory condition since it generates signatures using a superimposed coding. Finally, our access scheme can achieve very good retrieval performance because it can be easily adapted to a parallel execution environment.

Our trajectory access scheme consists of a partition table and a set of partitions. A partition can be divided into three areas; trajectory information, location information,

and signature information. A partition table maintains a set of partitions which store trajectories for current moving objects. The partition table is resided in a main memory due to its small size. To answer a user query, we find partitions to be accessed by searching the partition table. An entry  $E_i$  for a partition  $i$  is the following.

$E_i = \langle p\_start\_time, p\_end\_time, p\_expected\_time, final\_entry\_no \rangle$   
 -  $p\_start\_time, p\_current\_time, p\_end\_time$  : the smallest start time, the largest current and end time of all the trajectories in a partition  $I$ , respectively  
 -  $final\_entry\_no$  : the last entry number in a partition  $i$

Trajectory information area maintains moving object trajectories which consist of a set of segments (or edges). A trajectory  $T_i$  for an object  $MO_i$  is the following.

$T_i = \langle MOiid, \#\_past\_seg, \#\_future\_seg, \#\_mismatch, \{s_{ij}, eid, start, end, ts, (te \text{ or } v)\} \rangle$   
 -  $MOiid$  : ID for  $MO_i$   
 -  $\#\_past\_seg, \#\_future\_seg$ : the number of past and expected future segments  
 -  $\#\_mismatch$ : the # of mismatched segments between past and future segments  
 -  $s_{ij}$ :  $j$ -th segment of the trajectory for  $MO_i$ ,  $eid$ : edge ID for an edge covering  $s_{ij}$   
 -  $start, end$ : relative start and last location of  $s_{ij}$  in the edge of  $eid$ , respectively  
 -  $ts, te$ : start and end time of  $s_{ij}$  in the edge of  $eid$ , respectively  
 -  $v$  : average speed of  $s_{ij}$  in the edge of  $eid$  in case of a future segment

The location information area contains the location of an object trajectory stored in the trajectory information area. This allows for accessing the actual object trajectories corresponding to potential matches to satisfy a query trajectory in the signature information area. The location information area also allows for filtering out irrelevant object trajectories based on the time condition of a query trajectory because it includes the start time, the current time, and the end time for a set of object trajectories. A location information,  $I_i$ , for the trajectory of an object  $MO_i$  is the following.

$I_i = \langle MOiid, L_i, strat\_time, current\_time, end\_time \rangle$   
 -  $L_i$  : location for  $MO_i$  in the trajectory information area  
 -  $start\_time$  : time when the trajectory for  $MO_i$  is first inserted  
 -  $current\_time$  : time when the last segment of the trajectory for  $MO_i$  is inserted  
 -  $end\_time$  : time when the last expected segment for  $MO_i$  will be inserted

To construct our trajectory access scheme, it is necessary to create a signature from a given object trajectory in an efficient manner. To achieve it, we make use of a superimposed coding because it is very suitable to SNDB applications where the number of segments for an object trajectory is variable [ZMR98]. In case the total number of object trajectories is  $N$  and the average number of segments per object trajectory is  $r$ , optimal values for both the size of a signature in bits ( $S$ ) and the number of bits to be set per segment ( $k$ ) can be calculated.  $F_d$  is a false drop probability that a trajectory signature seems to qualify, given that the corresponding object trajectory does not actually qualify. Assuming that  $F_d$  is  $1/N$ , we can calculate the optimal values for  $S$  and  $k$  by using such formulas as  $\ln F_d = -(\ln 2)^2 * S/r$  and  $k = S * \ln 2/r$  [FC84]. In order that our signature-based access scheme should achieve good retrieval performance, we make use of a bit-sliced signature method for constructing our trajectory access scheme [ZMR98]. In the bit-sliced method, we create a fixed-length signature slice for each bit position in the original signature string. That is, we store a set of the

first bit positions of all the trajectory signatures into the first slice, a set of the second bit positions into the second slice and so on. In the bit-sliced method, when the number of segments in a query trajectory is  $m$  and the number of bits assigned to a segment is  $k$ , the number of page I/O accesses for answering the query in our signature-based access scheme is less than  $k*m$ . Our access scheme has a couple of advantages from the viewpoint of retrieval performance. First, when the number of segments in a query trajectory is small, our access scheme requires the small number of page I/O accesses due to the small number of signature slices needed for the query. Secondly, as the number of segments in a query trajectory is increased, the number of page I/Os to be accessed in the signature information area is increased, but the number of page I/O accesses in the trajectory information area is decreased. Thus, when the number of segments in a query trajectory is large, our signature-based access scheme achieves good retrieval performance eventually.

When a moving object trajectory is assumed to be hardly used for a trajectory-based query, the object trajectory should be moved to the past object trajectory structure (POTS) which is maintained in a tertiary storage. We make use of a strategy to move past object trajectories to POTS in a minimize cost by means of storing them into POTS with the unit of partition. In this case, the partition table maintains the information of all the partitions which will be answered for a trajectory-based query.

### 3.2 Insertion Algorithms

The algorithms for inserting moving objects trajectories can be divided into an initial trajectory insertion algorithm and a segment insertion algorithm for its trajectory. For the initial trajectory insertion, we find the last partition in the partition table and obtain an available entry (NE) in the last partition. To insert the initial trajectory with no expected trajectories, we create a new expected future segment based on an edge where an object currently moves and store it into the NE entry of the trajectory information area in the last partition. Using the expected future segment created, we store `start_time(StartT)`, `current_time(CurrentT)`, and `end_time(ExpectedET)` into the NE entry of the location information area in the last partition. Here `StartT` and `CurrentT` are both assigned to the start time of the moving object and `ExpectedET` is assigned to `NULL`. To insert the initial trajectory with expected trajectories, we insert a list of expected future segments (`TrajSegList`) into the NE entry of the trajectory information area in the last partition. We create a segment signature (`SSn`) from each of `TrajSegList` and generate a trajectory signature (`SigTS`) by using superimposing (`Oring`) all of the segment signatures. Using the `TrajSegList`, we store `StartT`, `CurrentT`, and `ExpectedET` into the NE entry of the location information area. `ExpectedET` is assigned to the expected end time of the last segment of the `TrajSegList`. Finally, we store the `SigTS` into the NE entry of the signature information area in the last partition, by using the bit-sliced manner. We store `<StartT, CurrentT, ExpectedET>` into the last partition entry (LP) of the partition table. Figure 2 shows the initial trajectory insertion algorithm (i.e., `InsertFirst`) for a moving object.

To insert the segment of a moving object trajectory, we find a partition storing its trajectory from the partition table by using the start time (ST) of the moving object. We obtain the entry (NE) storing the trajectory information of the partition, covering

```

Algorithm InsertFirst(MOid, TrajSegList)
/* TrajSegList contains the information of a set of expected segments for
the trajectory of a moving object MOid */
1. TrajSeg = the first segment of TrajSegList
2. Generate a signature SigTS from TrajSeg
3. StartT = CurrentT = ts of TrajSeg
4. Obtain final_entry_no of the entry, in the partition table, for the
last partition, LP
5. NE = final_entry + 1 //NE=the next available entry in LP
6. Obtain the location, Loc, of the entry NE in the trajectory info
area for inserting object trajectory
7. if(end field of TrajSeg=NULL){//no expected trajectory
8.   ExpectET = NULL
9.   Store <MOid,0,1,TrajSeg> into the entry NE, pointed by Loc, of
the trajectory information area in LP}
10. else { // expected trajectory exists
11.   #fseg = 1
12.   while (the next segment Sn of TrajSegList • NULL) {
13.     #fseg = #fseg + 1
14.     Generate a signature SSn from Sn
15.     SigTS = SigTS | SSn }
16.   Store <MOid,0,#fseg,TrajSegList> into the entry NE, pointed by
Loc,
of the trajectory info area in LP
17.   Compute ExpectET by using ts, start, and v of the last segment of
TrajSegList } // end of else
18. Store SigTS, using the bit-sliced manner, into the entry NE of the
signature information area in LP
19. Store <MOid,Loc,StartT,CurrentT,ExpectET> into the entry NE of the
location information area in the LP
20. Store <StartT,CurrentT,ExpectET,NE> into the entry for LP in the
partition table
End InsertFirst

```

---

**Fig. 2.** Initial trajectory insertion algorithm for moving objects

the object identified by MOid. To insert a segment for trajectories with no expected future ones, we store a new segment (TrajSeg) into the NE entry of the trajectory information area, being addressed by Loc. Then, we generate a trajectory signature (SigTS) from the TrajSeg and store the SigTS into the NE entry of the signature information area in the bit-sliced manner. Finally, we store <MOid, Loc, StartT, CurrentT, ExpectET> into the NE entry of the location information area. To insert a segment for trajectories with expected future ones, we can store a new segment according to three types of the discrepancy between a new segment and the expected segment of a trajectory by calling a function named find-seg(). First, in case of no segment coinciding with TrajSeg (seg\_pos = 0), we perform the same procedure as the segment insertion algorithm with no expected future segments. Secondly, in case where the segment coinciding with TrajSeg is the first one (seg\_pos = 1), we just store the TrajSeg into the (#\_actual\_seg)-th segment of the NE entry. Finally, in case where the segment coinciding with TrajSeg is not the first one (seg\_pos > 1), we delete seg\_pos-1 segments from the expected segments of the NE entry, store the TrajSeg into the (#\_actual\_seg)-th segment of the NE entry, and move all the expected segments of the NE entry forward by seg\_pos-2. If the ratio of mismatched segments (#\_mismatch) over all the segments of the trajectory is not greater than a threshold ( $\tau$ ), we store the trajectory signature (SigTS) generated from the TrajSeg into the NE entry of the signature information area in the bit-sliced manner. If the discrepancy is greater than  $\tau$ , we regenerate SigTS from the trajectory information

of the NE entry and store all the bits of the SigTS into all the bit-sliced signatures in the signature information area in the partition P. Finally, we update the values of #\_actual\_seg, #\_future\_seg, and #\_mismatch in the NE entry and update the CurrentT of the NE entry in the location information area as well as that of the partition P's entry in the partition table. Figure 3 shows the segment insertion algorithm (i.e., InsertSeg) for moving object trajectories.

---

```

Algorithm InsertSeg(MOid, TrajSeg, ST) /* TraSeg contains a segment for
the trajectory of a moving object MOid, to be stored with an object
trajectory's start time, ST*/
1. Generate a signature SigTS from TrajSeg
2. Locate a partition P covering ST in partition table
3. Locate an entry E covering ST for the moving object with MOid in
the location information area and get its location, Loc, in the
trajectory information area
4. Obtain #actual_seg, #future_seg, and #mismatch of the trajectory
info entry E (i.e., TE) for the MOid in P
5. if(#future_seg = 0) { // no expected trajectory
6.     Insert TrajSeg into the (#actual_seg+1)-th trajectory segment
of TE
7.     Store SigTS, using the bit-sliced manner, into the entry E of
the signature info area in P}
8. else { // expected trajectory exists
9.     seg_pos = find_seg(TrajSeg,Loc)
10.    #actual_seg++, #future_seg = #future_seg - seg_pos
11.    case(seg_pos = 0) { // find no segment
12.        Insert TrajSeg into segment of TE and relocate the future
traj segments backward
13.        Store SigTS, using the bit-sliced manner, into the entry E
of the signature info area in P }
14.    case(seg_pos = 1) //find the first segment
15.        Insert TrajSeg into (#actual_seg)-th trajectory segment of
TE for exchanging the old segment
16.    case(seg_pos > 1) { //find the (seg_pos)-th segment
17.        #mismatch = #mismatch + seg_pos - 1
18.        Insert TrajSeg into (#actual_seg)-th segment of TE and
relocate the future traj segments forward
19.        if(#mismatch/(#future_seg+#actual_seg) > •)
regenerate_sig(Loc,SigTS,E,P)} // end of case// end of else
20. Update #actual_seg, #future_seg, and #mismatch of TE
21. CurrentT = te of TrajSeg
22. Store CurrentT into the current_time of the entry E of the location
information area in the partition P
23. Store CurrentT into the p_current_time of the partition P entry in
the partition table
End InsertSeg

```

---

**Fig. 3.** Segment insertion algorithm for moving object trajectories

### 3.3 Retrieval Algorithm

The retrieval algorithm for moving object trajectories finds a set of objects whose trajectories match the segments of a query trajectory. To find a set of partitions satisfying the time interval (TimeRange) represented by <lower, upper> of a given query (Q), we call a find\_partition function to generate a list of partitions (partList) by performing the sequential search of the partition table and find a list of partitions (partList) to satisfy the following condition 3. Next, we generate a query signature

(QSig) from a query trajectory's segments. For each partition of the partList, we search only the signature slices corresponding to the bits set by '1' in QSig, in the signature information area. We create a list of candidates (CanList) being set to '1' by bit-oring the signature slices. For the entries corresponding to the candidates, we determine if their start\_time, end\_time, and current\_time satisfy the following condition 3. Finally, we determine if the query trajectory matches the object trajectories corresponding to the entries. The matching means that the object trajectory's segments contain those of the query trajectory. If the matching is found, we insert the object's ID into a result list (MOidList). Figure 4 shows the retrieval algorithm (i.e., Retrieve) for moving object trajectories.

$$\begin{aligned} &(\text{end\_time} \geq T.\text{lower}) \text{ AND } (\text{start\_time} \leq T.\text{upper}) \text{ if } \text{end\_time} \neq \text{NULL} \\ &(\text{current\_time} \geq T.\text{lower}) \text{ AND } (\text{start\_time} \leq T.\text{upper}) \text{ otherwise} \end{aligned} \quad (3)$$

---

```

Algorithm Retrieve(QSegList, TimeRange, MOidList) /* MOidList is a set
of ids of moving objects containing a set of query segments, QsegList,
for a given range time, TimeRange */
1. Qsig = 0, #qseg = 0, partList =  $\emptyset$ 
2. t1 = TimeRange.lower, t2 = TimeRange.upper
3. for each segment Qsj of QsegList {
4.   Generate a signature QSSI from Qsj
5.   QSig = QSig | QSSI, #qseg = #qseg + 1 }
   /*find partitions, partList, satisfying TimeRange by searching
   patiotion table of COTSS and B+-tree of POTSS*/
6. find_partition(TimeRange, partList)
7. for each partition Pn of partList {
8.   Obtain a set of candidate entries, CanList, by examining the
   bit slices of the signature info area in Pn, corresponding to
   bits set by 1 in QSig
9.   for each candidate entry Ek of CanList {
10.    Let s,e,c be start_time, end_time, current_time of the entry Ek
   of location information area
11.    if((s • t2) AND (e • t1 OR c • t1)){
12.      #matches = 0
13.      Obtain the first segment Esi of the entry Ek of the trajectory
   info area, TEk
14.      Obtain the first segment Qsj of QsegList
15.      while(Esi • NULL and Qsj • NULL) {
16.        if(match(Esi, Qsj)=FALSE)
           Obtain the next segment Esi of TEk
17.        else { #matches = #matches + 1
18.          Obtain the first segment Esi of TEk }
19.        if(#matches=#qseg)MOidList=MOidList  $\cup$  {TEk's MOid}
20.        } } //end of while //end of if //end of for- CanList
21. } // end of for - partList
End Retrieve

```

---

**Fig. 4.** Retrieval algorithm for moving object trajectories

## 4 Performance Analysis

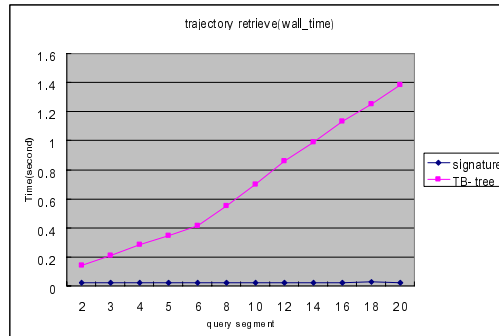
We implement our trajectory access scheme under Pentium-IV 2.0GHz CPU with 1GB main memory, running Window 2003. For our experiment, we use a road network consisting of 170,000 nodes and 220,000 edges [WMA]. For simplicity, we consider bidirectional edges; however, this does not affect our performance results. We also generate 50,000 moving objects randomly on the road network by using

Brinkhoff’s algorithm [B02]. For performance analysis, we compare our access scheme with TB-tree which is well known as an efficient trajectory access scheme for Euclidean spaces [PJT00], in terms of insertion time and retrieval time for moving object trajectories. First, Table 1 shows the insertion performance to store one moving object trajectory.

**Table 1.** Insertion performance

	TB-tree	Our signature-based scheme
Trajectory insertion time(ms)	0.03	0.95

It is shown from the result that our access scheme provides much worse insertion performance than TB-tree. The reason is because when a segment for a moving object trajectory is inserted, our access scheme needs to acquire a signature built for the moving object trajectory, change the signature, and store it in bit-sliced manner. On the contrary, TB-tree only needs to append the segment to the existing moving object trajectory.



**Fig. 5.** Retrieval performance

Next, we measure retrieval time for answering queries whose trajectory contains 2 to 20 segments. Figure 5 shows the retrieval time of our trajectory access scheme and TB-tree. It is shown from the result that our access scheme requires about 20 ms while TB-tree needs 140ms, when the number of segments in a query is 2. Thus our access scheme nearly 700% outperforms TB-tree. Furthermore, as the number of segments in queries increases, the retrieval time is increased in TB-tree; however, our access scheme requires constant retrieval time. When the number of segments in a query is 20, it is shown that our access scheme requires about 24 ms while TB-tree needs 1380ms. Thus our access scheme achieves about two orders of magnitude better retrieval performance than TB-tree. The reason is why our trajectory access scheme creates a query signature combining all the segments in a query and searches for potentially relevant trajectories of moving objects once by using the query signature as a filter because it generates signatures based on a superimposed coding technique. On the contrary, TB-tree achieves bad performance due to a large extent of overlap in its internal nodes when the

number of segments in a query trajectory is small. TB-tree doesn't achieve good retrieval performance due to a large volume of dead spaces in its nodes when the number of segments in a query trajectory is large. In addition, TB-tree builds a MBR for each segment in a query and performs a range search for each MBR. Because the number of range searches increases in proportion to the number of segments, TB-tree dramatically degrades on trajectory retrieval performance when the number of segments is great.

## 5 Conclusions

Because moving objects usually moves on spatial networks, instead of on Euclidean spaces, efficient index structures are needed to gain good retrieval performance on their trajectories. However, there has been little research on trajectory access schemes for spatial network databases. Therefore, we proposed an efficient trajectory access scheme for indexing moving objects. For this, we designed our access scheme for efficiently dealing with the trajectories of moving objects on road networks. In addition, we provided an initial trajectory insertion algorithm as well as a segment insertion one, and we provided a retrieval algorithm to find a set of moving objects whose trajectories match the segments of a query trajectory. Finally, we show that our trajectory access scheme achieves about one order of magnitude better retrieval performance than TB-tree. As future work, it is needed to extend our access scheme to a parallel environment so as to achieve better retrieval performance because its parallel execution can be performed easily due to the characteristic of signature files [ZMR98].

## References

- [B02] T. Brinkhoff, "A Framework for Generating Network-Based Moving Objects," *GeoInformatica*, Vol. 6, No. 2, pp 153-180, 2002.
- [F03] E. Frenzos, "Indexing Objects Moving on Fixed Networks," *Proc. of SSTD*, pp 289-305, 2003.
- [FC84] C. Faloutsos and S. Christodoulakis, "Signature Files: An Access Method for Documents and Its Analytical performance Evaluation," *ACM Tran. on Office Information Systems*, Vol. 2, No. 4, pp 267-288, 1984.
- [HKT02] M. Hadjieleftheriou, G. Kollios, V.J. Tsotras, and D. Gunopulos, "Efficient Indexing of Spatiotemporal Objects," *Proc. of EDBT*, pp 251-268, 2002.
- [PJT00] D. Pfoser, C.S. Jensen, and Y. Theodoridis, "Novel Approach to the Indexing of Moving Object Trajectories," *Proc. of VLDB*, pp 395-406, 2000.
- [PZM03] S. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query Processing in Spatial Network Databases," *Proc. of VLDB*, pp, 802-813, 2003.
- [Se99] S. Shekhar et al., "Spatial Databases - Accomplishments and Research Needs," *IEEE Tran. on Knowledge and Data Engineering*, Vol. 11, No. 1, pp 45-55, 1999.
- [SKS03] C. Shahabi, M.R. Kolahdouzan, M. Sharifzadeh, "A Road Network Embedding Technique for K-Nearest Neighbor Search in Moving Object Databases," *GeoInformatica*, Vol. 7, No. 3., pp 255-273, 2003.
- [SJK03] L. Speicys, C.S. Jensen, and A. Kligys, "Computational Data Modeling for Network-Constrained Moving Objects," *Proc. of ACM GIS*, pp 118-125, 2003.
- [WMA] <http://www.maproom.psu.edu/dcw/>
- [ZMR98] J. Zobel, A. Moffat, and K. Ramamohanarao, "Inverted Files Versus Signature Files for Text Indexing," *ACM Tran. on Database Systems*, Vol. 23, No. 4, pp 453-490, 1998.