A Pattern-based Predictive Indexing Method for Distributed Trajectory Databases

Keisuke KATSUDA[1⋆], Yutaka YANAGISAWA[2], and Tetsuji SATOH[1,2]

[1] Graduate School of Information Science and Technology, Osaka University, Japan.
[2] NTT Communication Science Laboratories, NTT Corporation, Japan.

Abstract. Recently, it has become possible to collect large amounts of trajectory data of moving objects by using sensor networks. To manage such trajectory data, we have developed a distributed trajectory database composed of a server and many sensor nodes deployed over wide areas. The server manages the trajectory data of each moving object by using indices. However, since each sensor node cannot send trajectory data to the server all the time, the server does not always manage indices for the current trajectory data. In other words, the server is delayed in answering queries for current data because it has to forward each query to the sensor nodes to answer them. This is defined as a *delay problem*. To avoid this problem, we propose a pattern-based predictive indexing method for the database to answer queries in real time. This method uses past motion patterns of moving objects to predict the future locations of moving objects. In this paper, we describe the method and evaluate it with practical trajectory data. We conclude that the technique can predict the future locations of moving objects well enough in real time and show optimal parameters for prediction.

1 Introduction

In recent years, various types of applications using the trajectory data of moving objects have been developed [1] [2] and have attracted attention because they allow us to obtain high accurate trajectories using positioning devices on sensor networks. Applications include forecasting traffic congestion, management of taxis and trucks, automatic switching of point-of-purchase advertisements, and so on. These systems must deal efficiently with a large amount of trajectory data (see Fig. 1). However, since the amount of trajectory data has been growing rapidly year by year and such data are managed over wide areas, it is difficult to manage all it in a single database [3].

Therefore, we have developed a distributed trajectory database (DTDB) that stores trajectory data in distributed environments as sensor networks. DTDB consists of a server and many sensor nodes connected to that server. Each sensor node has a positioning device and a database that stores the obtained position data. Nodes do not send all of the trajectory data to the server but only the

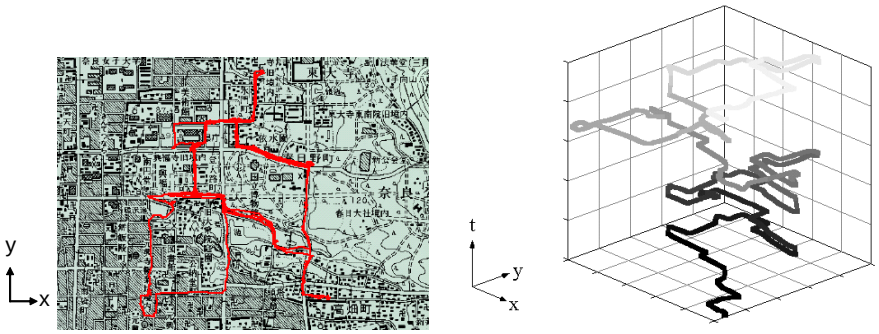
⋆ corresponding author: k-katuda@ist.osaka-u.ac.jp

data necessary for the server to generate indices. Using the indices, the server can answer a *window query* to find the objects intersecting a query window during a past time interval, even though the server does not store all of the obtained trajectory data.

However, there is a problem associated with distributing data. Since each sensor node does not send the trajectory data to the server in real time, the server may have to wait for the data from the sensors to generate indices, which are used to answer window queries. Therefore, the server may answer the queries late. We call this the *delay problem*. To avoid this problem, the server must predict future trajectory data and generate predictive indices corresponding to future trajectory data. Using predictive indices, the server can answer a *future window query* to find objects intersecting a query window not only during past time but future time.

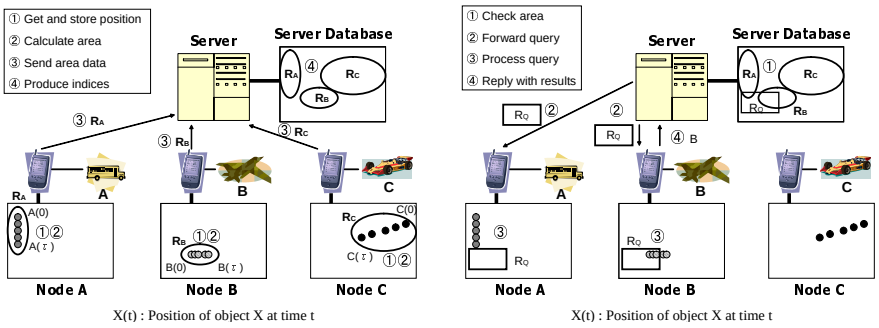
We propose a pattern-based predictive indexing method for the future position of moving objects. In this paper, we describe a method that uses the past motion patterns of moving objects extracted from past trajectories. Moreover, we develop a DTDB prototype to evaluate our proposed method with practical trajectory data on rickshaws in Nara, Japan. In this evaluation, we investigate the effects of variations in the length of trajectory data for prediction, the data granularity, and the transmission interval of sensor nodes on the prediction.

The rest of the paper is organized as follows. In Section 2, we describe DTDB and the delay problem in detail. Section 3 describes our proposed method. In Section 4, we evaluate our method with comprehensive experiments using trajectory data from rickshaws. Section 5 introduces related work and explains differences from our method. Finally, Section 6 concludes the paper with a discussion of future work.



(a) x-y plane (b) x-y-t space

Fig. 1. Sample trajectory data of a rickshaw



(a) Generating indices (b) Processing a query

Fig. 2. Distributed trajectory database

2 Distributed Trajectory Database

2.1 Overview

We define trajectory data as the sequence of both the position and the time of a moving object. Fig. 1(a) shows the trajectory of a rickshaw moving around Nara. In Fig. 1(b), the same trajectory is projected in x-y-t space.

We consider a trajectory database that comprises both a server database and many positioning devices embedded within a moving object [4]. When each positioning device obtains the location of an object, it sends the data to the server. In other words, the server collects all the trajectory data of all objects to answer a *window query* [5]. Thus, the database is a system that is effective enough to answer queries for moving objects in real time.

However, since sensors are becoming cheaper and smaller and sensor networks are growing, in the future it will become more difficult to manage all trajectory data at a single location. Therefore, we have developed a distributed trajectory database, a distributed version of a trajectory database.

DTDB comprises a server and many sensor nodes connected to the server. Each sensor node has both a positioning device and a database. The former is embedded within a moving object and stores obtained trajectory data in its embedded database. Fig. 2(a) illustrates the process by which the server database generates indices to the data stored at the embedded databases. In the example, there are three sensor nodes: A, B, and C. Each node obtains the position of a moving object and stores in its database at each time interval and calculates the maximum area within which an object moves at regular interval τ. Moreover, each sensor node sends area $\Re$ to the server at interval τ. On the other hand, the server database generates indices from received area $\Re$. Each index at $t = \tau$ indicates the area to which the object moved within $0 \leq t \leq \tau$. In Fig. 2(a), R_A,

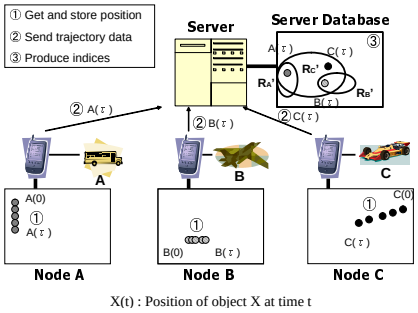


Fig. 3. Producing predictive indices

R_B, and R_C are the areas within which objects A, B, and C respectively move $0 \leq t \leq \tau$.

Next, we illustrate the process by which the server database retrieves the data indicated by a given query window. A query is also given as area R_Q in Fig. 2(b). When the server receives a query, it verifies whether R_Q overlaps the areas stored in the server database. If R_A and R_B overlap with R_Q, the server forwards R_Q to sensor nodes A and B. When the sensor nodes receive a query, they process it in their embedded databases. Finally, each sensor node receiving a query replies with the results of the query to the server. In this case, the result of R_Q is B. In this process, the server can efficiently retrieve any object by using indices.

2.2 Delay Problem and Approach

In this section, we describe the *delay problem* that occurs in DTDB. Before describing it, we state two assumptions.

- The server receives data and generates indices to the data ($(n-1)\tau \leq t \leq n\tau$) at $t = n\tau$ ($n \in \mathbb{N}$), where $\mathbb{N}$ is the set of all natural numbers.
- The server manages the indices to the data ($0 \leq t \leq n\tau$) at $t = n\tau$.

Therefore, at $t = n\tau + j$, where $j \in \mathbb{N}$ and $j < \tau$, the server can search for the data at $t \leq n\tau$. If the server searches for the data at $t = n\tau + j$, it has to wait until $t = (n+1)\tau$, when indices corresponding to the data at $t = n\tau + j$ are generated. As a result, a delay of $\tau - j$ occurs. We call this the *delay problem* in DTDB.

To avoid this problem, we introduce a method that predicts data from $n\tau+1$ to $(n+1)\tau-1$, using already received data. By applying this prediction technique, the server produces predictive indices to answer queries for data that have not been received yet.

We show the process by which the server produces predictive indices in Fig. 3. In this example, sensor nodes A, B, and C obtain their position at every time

Table 1. Definition of symbols

X_t	position of X at t ($X_t = (x, y)$)
S_X	CID sequence of X in ascending time
$S_X(n)$	n th CID of S_X
$\|S_X\|$	element number of S_X
L	element number of CID sequence for prediction

interval, and each node stores trajectory data in their embedded databases. They also send trajectory data to the server at every regular interval τ. At $t = n\tau$, the server generates indices corresponding to the data ($n\tau < t \leq (n+1)\tau$) using the past trajectory data at the time when the server received the trajectory data. Suppose that now $t = \tau$; the three circles R'_A, R'_B, and R'_C are the areas within which each object will move $\tau < t \leq 2\tau$. The server database uses these predictive indices to answer a query for $\tau < t \leq 2\tau$.

To produce a predictive index, the server must predict the positions at which an object will be in the future. We describe the proposed prediction technique to calculate the future positions of moving objects in Section 3 and evaluate the technique in Section 4.

3 Pattern-based Predictive Indexing Method

In this section, we describe our pattern-based predictive indexing method, which assumes that an object tends to move along the trajectories of other moving objects. Based on this assumption, the positions of objects can be predicted by using motion patterns extracted from the trajectories of other moving objects.

In the following we explain the process for extracting motion patterns from trajectories. First, the server divides the entire area into a grid with several small cells; each cell has a cell identification label (CID). After receiving the trajectory data from sensors, the server records the CID at the point where each object enters. The server manages the CID sequences as the motion patterns of moving objects. We define the notation to describe how our method predicts future locations of moving objects in Table 1.

For predicting the future positions of moving object X, the server compares the last several CID sequences of X with all stored past CID sequences of all objects. The server obtains the CID subsequence most similar to the sequence of X by comparing CID sequences of every object with S_X. As a result, the server uses the next CID of the obtained CID subsequence as a cell to which X will move in the future. In Fig. 4, we show an algorithm that predicts the most probable cell to which an object will move in the future.

Fig. 5 shows an example of the prediction technique. In Fig. 5(a), object X moves around a grid divided into 9 cells. The small circles show the positions of object X at $t = cn$, $c, n \in \mathbb{N}$, and c const. The numbers from 00 to 22 indicate CIDs. The server manages the CID sequence of X as $S_X = \langle 00, 10, 20, 21, 22, 12, 02, 01 \rangle$. In Fig. 5(b), there are two positions of X: at $t =$

```
INPUT : CID sequence SET S = {S1,…,Sn} / * CIDs of all objects * /
        CID sequence Sq / * CIDs to be predicted * /
        CID SET Θ = {θ1,...,θm}
        int L / * |Sq| * /
OUTPUT : CID c / * the most probable cell where object q will move * /
CID Prediction(S,Sq,Θ){
 int Ps[m] (for each s in S), i, j, k
   for each s in S{
     for(i = |s|;i ≥ L;i − −){
       if(Sq(L) == s(i))then {
         Ps[s(i +1)] = i;
         j = i −1;
         for(k = L −1;k ≥ L;k − −){
           if(Sq(k) == s( j))then
             Ps[s(i +1)] = Ps[s(i +1)]+ j;
           j = j −1} } }
 if(all Ps == 0)then / * there is no similar trajectory * /
   return Sq(L); / * the same cell as predecessor * /
 else
   return c where maximum Ps[θs]  (for each s in S, θ in Θ);}
```

Fig. 4. Algorithm obtaining most probable area

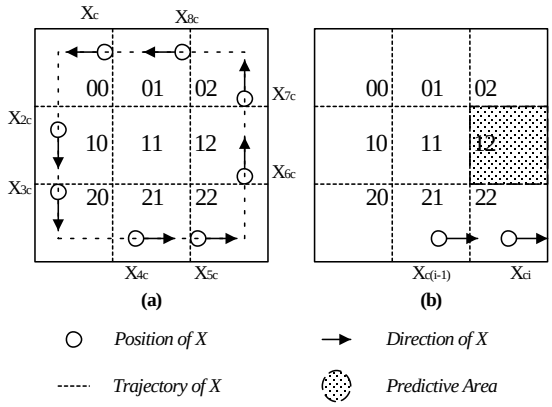


Fig. 5. Example of indexing moving objects

$c(i-1)$ and $t = ci$ $(i \in \mathbb{N})$. Suppose that $t = ci$, then the CID sequence to be compared is $S_q = \langle 21, 22 \rangle$ if $L = 2$. The server calculates that X will move to the area of $\mathrm{CID}(S_X(6) = 12)$ at $t = c(i+1)$ by using that algorithm (Fig. 4).

4 Performance Study

In this section, we describe experiments conducted to evaluate the performance of our proposed method for a DTDB, using a DTDB prototype system.

Table 2. Defenitions of evaluation symbols

P_{inter}	transmission interval (second)
P_{len}	length of a CID sequence for prediction
P_{grid}	number of cells in the grid
Ω	amount of trajectory data (byte)
ω	amount of predictable trajectory data (byte)
P_{acc}	predictive accuracy ($= \omega/\Omega$)

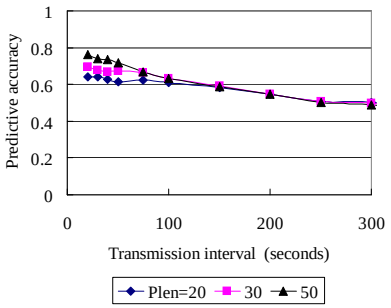
4.1 Settings

The system comprises of many embedded databases equipped with GPS and a server database. Each sensor obtains its position at any time and manages the data in its own embedded database. Every embedded database sends data to the server database at regular intervals. The server database generates indices corresponding to the data stored at embedded databases using the received data. It can also deal with *future window queries* from users using the indices. First, the server database identifies the embedded database managing the data that will be used for the query results. Second, the server forwards the query to all identified sensors. After receiving the query, the sensors process it in their database and send the results to the server. Consequently, query results can be answered.

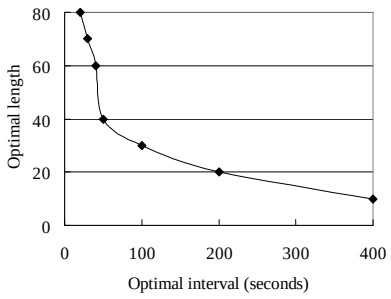
In this evaluation we use the trajectory data from 10 rickshaws during 4 days in the city of Nara, Japan. Each trajectory data has 20,000 position data values at 20,000 seconds. The notation used for the evaluation is shown in Table 2. The system targeted in this paper is assumed to be an application operating with sensor networks composed of many battery-powered sensors. Therefore, sensors must reduce the number of transmissions and the amount of transmitted data. It is important to predict as much data as possible in the most effective manner. Therefore, it is desirable to obtain high predictive accuracy at large transmission intervals, using only a little information for prediction and grids that are divided into as many cells as possible. Consequently, we compare P_{acc} by varying P_{inter}, P_{len}, and P_{grid}.

4.2 Results

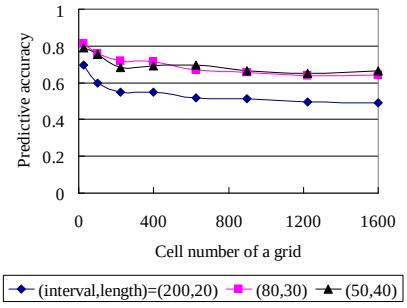
We show simulation results in Fig. 6 by plotting average values taken from four days of simulations in each scenario. Fig. 6(a) shows P_{acc} for P_{inter} under three different P_{len} (P_{grid} is kept constant at 30×30). The figure indicates that P_{acc} generally tends to increase while P_{inter} decreases. Moreover, P_{acc} tends to increase with P_{len} for $P_{inter} < 200$. On the contrary, P_{acc} does not increase even if P_{len} increases where $P_{inter} > 200$. These results suggest that there is a limit at which P_{acc} does not increase any more, regardless of increases of P_{len} in each P_{inter}. For example, when $P_{inter} = 200$, the limit is $P_{acc} = 0.55$, and the minimum length of CID sequences for prediction is $P_{len} = 20$. We call P_{inter}, P_{len} at such a limit point *optimal interval* and *optimal length*, as shown



(a) P_{acc} vs P_{inter}



(b) Optimal length vs optimal interval



(c) P_{acc} vs P_{grid}

Fig. 6. Experiment results

in Fig. 6(b), the optimal length decreases exponentially with increasing optimal intervals. Thus, changes of the optimal length are large for small optimal interval (< 50) and small for large optimal intervals (> 200).

Moreover, we experimentally evaluate the effects of P_{grid} on P_{acc}. Fig. 6(c) shows P_{acc} for P_{grid} under three different groups of optimal intervals and lengths: (200,20), (80,30), (50,40). If the server divides the grid into many cells, the areas of each cell are small, confining the predictive area to a small area. Fig. 6(c) indicates that P_{acc} decreases at most by 20% if P_{grid} increases to $P_{grid} = 1,600$. Therefore, we can increase the number of cells to a large number in the system without requiring high predictive accuracy.

Consequently, we can obtain an effective system by using these results. For example, where $P_{inter} = 50$, $P_{len} = 40$, and $P_{grid} = 1,600$, our proposed method

can obtain about 70% predictive accuracy, and the predictive area is confined to $1/1,600$ of the entire area.

5 Related Work

In this section, we give an overview of related work and show the advantages of our method by comparing it with other approaches. Many publications related to our proposed method have attempted to retrieve moving objects from a database system.

Modern database applications dealing with moving objects are usually managed by a *spatial-temporal database management system* (STDBMS). Recently, STDBMS research has attracted a great deal of attention [5] [6] [7] [8] [9] [10] [11] [12] [13]. In STDBMS, the location of a moving object is represented as a function of time, and the database stores such function parameters as velocity and location. The system is updated only when an object changes any of its moving parameters. To manage the locations of moving objects, many indexing methods have been proposed [8] [10] [11] [12] [13]. However, in STDBMS the sensor nodes must send parameters to the server whenever they change. Since such moving objects as people and cars rarely go straight for a long while, sensor nodes have to send parameters to the server frequently. On the contrary, in our database (DTDB), sensor nodes must send trajectory data to the server at constant intervals, however they need not send so frequently.

Also, several papers describe predictive indexing methods [10] [11] [12] [13] that process future queries in moving object or trajectory databases. These indexing methods predict the future locations of moving objects using only positions or velocities of objects. Therefore, these methods cannot predict the locations of moving objects that continually turn. However, since our indexing method predicts the future locations of moving objects using the past trajectory patterns of moving objects, our method can also predict the locations of moving objects that continually turn.

6 Conclusion

In this paper, we proposed a pattern-based predictive indexing method for DTDB and evaluated it by using a prototype DTDB system. As a result of the experiments, we obtained optimal values of both transmission intervals of sensor nodes and the length of trajectory data for prediction.

We have every confidence that our proposed method will locate moving objects well in real time. Currently, we are planning to incorporate trajectory data from such additional objects as cars and pedestrians. We are also investigating other predictive indexing methods using destination and purpose of moving.

References

1. Laube, P., Imfeld, S.: Analyzing relative motion within groups of trackable moving point objects. In: Proceedings of GIScience 2002 Conference, Boulder, CO, USA, Springer-Verlag Heidelberg (2002) 132–144
2. Vazirgiannis, M., Wolfson, O.: A spatio temporal model and language for moving objects on road networks. In Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J., eds.: Proceedings of SSTD 2001. Volume 2121 of Lecture Notes in Computer Science., Springer-Verlag (2001) 20–35
3. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: Proceedings of the 29th VLDB Conference, Berlin, German (2003) 12–23
4. Yanagisawa, Y., Akahani, J., Satoh, T.: Shape-based similarity query for trajectory of mobile objects. In: Proceedings of the 4th International Conference on Mobile Data Management, Melbourne, Australia (2003) 63–77
5. Saltenis, S., S.Jensen, C., T.Leutenegger, S., Lopez, M.A.: Indexing the positions of continuously moving objects. In: Proceedings of SIGMOD Conference. (2000) 331–342
6. Guttman, O.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of SIGMOD'84 Conference. (1984) 47–57
7. Zhang, Q., Lin, X.: Clustering moving objects for spatio-temporal selectivity estimation. In: Proceedings of the fifteenth conference on Australasian database. Volume 27. (2004) 123–130
8. Kollios, G., Tsotras, V.J., Gunopulos, D., Delis, A., Hadjieleftheriou, M.: Indexing animated objects using spatiotemporal access methods. Knowledge and Data Engineering **13** (2001) 758–777
9. Agarwal, P.K., Arge, L., Erickson, J.: Indexing moving points. In: Proceedings of Symposium on Principles of Database Systems. (2000) 175–186
10. Tao, Y., Sun, J., Papadias, D.: Selectivity estimation for predictive spatio-temporal queries. In: Proceedings of International Conference on Data Engineering. (2003) 417–428
11. Choi, Y.J., Chung, C.W.: Selectivity estimation for spatio-temporal queries to moving objects. In: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, USA, ACM SIGMOD international conference on Management of data table of contents, ACM Press (2002) 440–451
12. Hadjieleftheriou, M., Kollios, G., Tsotras, V., Gunopulos, D.: On-line discovery of dense areas in spatio-temporal databases. In: Proceedings of the 8th SSTD Conference, Santorini, Greece (2003) 306–324
13. Hadjieleftheriou, M., Kollios, G., Tsotras, V.J.: Performance evaluation of spatio-temporal selectivity estimation techniques. In: Proceedings of 15th International Conference on Scientific and Statistical Database Management, Cambridge, Massachusetts, USA, IEEE Computer Society (2003) 202–211