# A Hybrid Prediction Model for Moving Objects

Hoyoung Jeung† , Qing Liu‡, Heng Tao Shen†, Xiaofang Zhou†

† *The University of Queensland, National ICT Australia (NICTA), Brisbane*
{hoyoung, shenht, zxf}@itee.uq.edu.au
‡ *Tasmanian ICT Centre, CSIRO, Australia*
q.liu@csiro.au

***Abstract*— Existing prediction methods in moving objects databases cannot forecast locations accurately if the query time is far away from the current time. Even for near future prediction, most techniques assume the trajectory of an object's movements can be represented by some mathematical formulas of motion functions based on its recent movements. However, an object's movements are more complicated than what the mathematical formulas can represent. Prediction based on an object's trajectory patterns is a powerful way and has been investigated by several work. But their main interest is how to discover the patterns. In this paper, we present a novel prediction approach, namely *The Hybrid Prediction Model*, which estimates an object's future locations based on its pattern information as well as existing motion functions using the object's recent movements. Specifically, an object's trajectory patterns which have ad-hoc forms for prediction are discovered and then indexed by a novel access method for efficient query processing. In addition, two query processing techniques that can provide accurate results for both near and distant time predictive queries are presented. Our extensive experiments demonstrate that proposed techniques are more accurate and efficient than existing forecasting schemes.**

## I. Introduction

Given an object's recent movements and the current time, predictive queries ask for the object's probable location at some future time. Most existing techniques cannot predict accurately when the query time is far away from the current time. This is because all of these prediction methods are based on the object's recent movements, which may not be of much assistance for a distant time prediction. For example, even if we know Jane was at home at 9:00 a.m. and she is passing by a shopping center currently (9:05 a.m.), it is unreasonable to predict what her location will be at noon based on the above movements. We claim that an object's recent movements are only helpful to predict near future locations.

Furthermore, even for the near future prediction, most techniques assume the predictive trajectories of an object can be represented by some mathematical formulas based on its recent movements. In real world, however, an object's movements are more complicated than what the mathematical formulas can represent. Their movements may be affected by road networks and traffic jams for vehicles, turbulence places for aircraft, and so on. Fig. I shows an example in which Jane drives to work along the solid line on weekday avoiding a traffic jam. Existing approaches using linear motion functions [1], [2], [3], [4], [5] may fail to predict the location at time $t_2$ since the functions will return position $F_1$. Although [6] can capture nonlinear motions, it may also give an incorrect prediction $F_2$ at time $t_6$.

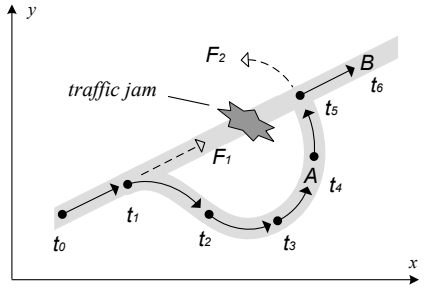



Fig. 1. Inadequate Prediction

In fact, an object's movements follow some patterns in many applications. People go to work every weekday along similar routes, public transportation is governed by time schedules and destinations, and animals annually migrate to reproduce or seek warmer climates. These patterns can provide reasonable predictions if they satisfy some query conditions. In Fig. I, for example, if we know that the object frequently visits $B$ at $t_6$ after $A$ at $t_4$ and it is currently passing $A$ at $t_4$, we can say the object is likely to be on $B$ at $t_6$ instead of $F_2$.

The problem of discovering an object's trajectory patterns has been discussed in several work. Although some of them discover patterns for forecasting an object's future positions [7], [8], [9], their major interest is about how to find the patterns over the object's historical movements. They assume the queries could be answered easily by the discovered patterns. However, this is not always true. In order to obtain the patterns, a large volume of an object's historical movements is required for a data mining process. It also implies that the number of patterns discovered could be very large. Therefore, the method of organizing these patterns to answer the predictive queries efficiently is important and needs to be addressed. Nevertheless, none of the studies has shown an effective management scheme for a large number of patterns. Though [10] introduces some pattern indexing techniques, it focuses on historical queries. Further more, none of them could sensibly answer the predictive queries in the situation when no pattern was available.

In this paper, we address the problem of how to discover trajectory patterns and how to manage a large number of trajectory patterns to answer predictive query efficiently. Moreover, we present a reasonable way to answer the distant time predictive query in order to overcome the shortcomings of current state-of-the-art techniques. To the best of our knowledge, this is the first work investigating this problem.

Our extensive experiments demonstrate that predictive queries can be answered accurately as well as efficiently. The main contributions of this paper are summarized below.

- We define the concept of *trajectory pattern* which is especially designed for predictive queries and its similarity measures.
- We present a novel data access method, *Trajectory Pattern Tree* (TPT), which indexes the trajectory patterns to answer predictive queries efficiently.
- We propose a *Hybrid Prediction Algorithm* (HPA) that provides accurate predictions for both near and distant time queries.
- We present comprehensive experimental results over various datasets. The results demonstrate that our techniques are more accurate and efficient than existing forecasting schemes.

The rest of the paper is organized as follows: In Section II, we provide preliminaries and related work. Section III gives an overview of the proposed framework. How to discover trajectory patterns and index them are presented in Section IV and V respectively. Hybrid prediction algorithms are shown in Section VI. Section VII discusses comprehensive experimental evaluations and Section VIII concludes.

## II. Related Work

### A. Vector Based Prediction

In the past several years, predictive query processing has been paid a great amount of attention by the spatio-temporal database society [1], [2], [3], [11]. For efficient query processing, various access methods have been proposed such as the *time-parameterized* method [4] and its variations [5], [6], and *dual transformation* techniques [11], [2]. Despite of the variety of index structures, all of them estimated objects' future locations by motion functions.

The motion functions can be divided into two types: (1) *linear models* that assume an object follows linear movements [4], [5], [3], [2] and (2) *non-linear models* that consider not only linearity but also non-linear motions [6], [12]. Given an object's location $l_0$ at time $t_0$ and its velocity $v_0$, the linear models estimate the object's future location at time $t_q$ by using the formula $l(t_q) = l_0 + v_0 \times (t_q - t_0)$, where $l$ and $v$ are $d$-dimensional vectors. The non-linear models capture the object's movements by more sophisticate mathematical formulas. Thus, their prediction accuracies are higher than those of the linear models.

*Recursive Motion Function* (RMF) [6], is the most accurate prediction method among both types of motion functions in the literature. It formulates an object's location at time $t$ as $l_t = \sum_{i=1}^{f} c_i \cdot l_{t-i}$, where $c_i$ is a constant matrix and $f$ (called $retrospect$) is the minimum number of the most recent timestamps which are needed to compute the elements of all $c_i$. In spite of many outstanding features of RMF, it has a couple of limitations. First, RMF is useful for only near future predictions. When a query time is distant from the current time, prediction accuracies may decrease dramatically. Second, for the near future predictions, it still cannot capture sudden changes of the object's velocities (e.g., a cars left-turn or u-turn) as the function is only influenced by previous locations.

### B. Pattern Based Prediction

*Discrete Markov models* [13] have been used for estimating the future locations of an object among spatial cells. [8], [14] derive the Markov transition probabilities from one or multiple cells to another. They then look at to which cell the object belongs currently and compute the next cell in which the object is likely to be in future, based on the Markov models.

*Association rules* are also utilized for location predictions. [15], [16], [7] address *spatio-temporal association rules* of the form $(r_i, t_1, c) \longrightarrow (r_j, t_2)$ with a confidence $c$, where $r_i$ and $r_j$ are regions at time $t_1$ and $t_2$ respectively $(t_2 > t_1)$. It implies that an object in $r_i$ at time $t_1$ is likely to appear in $r_j$ at time $t_2$ with $c$ probability.

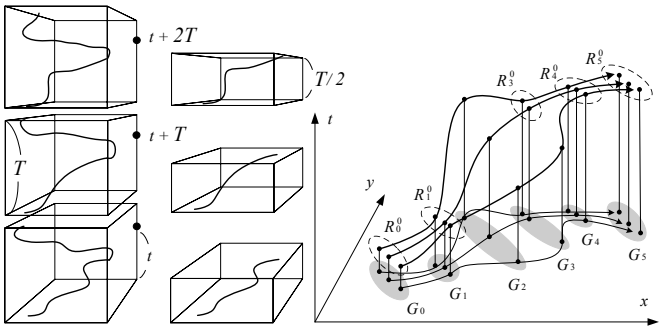
[9], [17] define and mine sequential patterns of an object's trajectories. Though the work in [9] is very close to our study in terms of hybrid prediction, it mainly focuses on developing discovery techniques rather than utilizing patterns for prediction.

All of the above techniques have some common deficiencies for moving object prediction. First, those methods except [9] cannot give a reasonable location when there is no pattern at a given query time. For example, [7] picks one neighbor cell randomly when no pattern is available. Certainly, this approach cannot give a precise answer. Second, prediction accuracies in the studies are considerably affected by the size of each cell in data space. Nevertheless, none of the studies shows an efficient space management technique. Third, those techniques can discover a large number of trajectory patterns. However, they do not consider how to manage the large amount of patterns or how to search the specific patterns matching given queries quickly.

## III. Hybrid Prediction Model

An object's trajectory is typically represented as a sequence $\{(l_0, l_1, ..., l_i, ..., l_{n-1})\}$, where $l_i (0 \leq i < n)$ denotes the object is at location $l$ at time $i$. Likewise [10], we consider discovering an object's periodic patterns from its historical trajectory. Given $T$, which is the number of timestamps that a pattern may re-appear, an object's trajectory is decomposed into $\lfloor \frac{n}{T} \rfloor$ sub-trajectories (Fig. 2(a)). $T$ is data-dependent and has no definite value. For example, $T$ can be set to 'a day' in traffic control applications since many vehicles have daily patterns, while the behaviors of animals' annual migration can be discovered by $T =$ 'a year'.

All locations from $\lfloor \frac{n}{T} \rfloor$ sub-trajectories which have the same time offset $t$ of $T$ $(0 \leq t < T)$ will be gathered onto one group $G_t$. $G_t$ represents all locations that the object has appeared at time offset $t$. A clustering method is then applied to find dense clusters $R_t$ in each $G_t$. Fig. 2(b) shows an example of the above concepts. $R_t$ symbolizes the region inside of which the object may often appear at time offset $t$. We call $R_t$ a




(a) Trajectory decomposition (b) Projected sub-trajectories

Fig. 2. Periodic Pattern Discovery

*frequent region* at $t$. More than one frequent region at time offset $t$ can exist. For example, Jane leaves home at 9:00 a.m. and passes through the city at 10:00 a.m. to go to her work every weekday. On most weekends, she passes by a shopping center at 10:00 a.m. on the way to the beach. In this case, there are two frequent regions at 10:00 a.m.. To distinguish these frequent regions having the same time offset, we use $R_t^j$ to represent the $j^{th}$ frequent region at time offset $t$.

Consider an example in Fig. 3, which Jane is at home ($R_0^0$) at time offset 0 and then in the city ($R_1^0$) at time offset 1. By examining her movement history, we can derive the probability that she would be at her work place ($R_2^0$) at time offset 2 is 0.5. We use an association rule $R_0^0 \wedge R_1^0 \xrightarrow{0.5} R_2^0$ to represent the above knowledge. On the other hand, if she passes the shopping center ($R_1^1$) at time offset 1 instead of the city, she will be at the beach ($R_2^1$) with probability 0.4. This is represented as $R_0^0 \wedge R_1^1 \xrightarrow{0.4} R_2^1$. The concept of a *trajectory pattern* is then formally defined as follows:

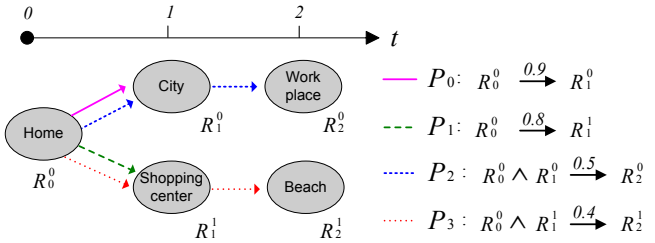



Fig. 3. An Example of Trajectory Patterns

*Definition 1:* A **trajectory pattern** $P$ is a special association rule of the form $R_{t_1}^{j_1} \wedge R_{t_2}^{j_2} \wedge ... \wedge R_{t_m}^{j_m} \xrightarrow{c} R_{t_n}^{j_n}$ with time constraint $t_1 < t_2 < ... < t_m < t_n$.

We call $R_{t_1}^{j_1} \wedge R_{t_2}^{j_2} \wedge ... \wedge R_{t_m}^{j_m}$ the *premise* and $R_{t_n}^{j_n}$ the *consequence*. *Confidence* $c$ means that when the premise occurs, the consequence will also occur with probability $c$.

Given an object's recent locations, the current time $t_c$ and the query time $t_q$, a predictive query estimates the object's future position at $t_q$. In this paper, we aim to answer the predictive query using not only the object's recent movements but also its trajectory patterns because the recent movements may not be sufficient for prediction in many cases. Assume Jane's location at 10:00 a.m. may be predicted by her recent locations at time 9:50 a.m. and 9:55 a.m., but it is unpredictable if the query time is 5:00 p.m., let alone if the query time is 5:00 p.m. tomorrow. It means the prediction accuracy will be dramatically decreased when the time distance between the query time and the current time increases. Because the time distance has an impact on the prediction accuracy, we distinguish predictive queries having long prediction length in time.

*Definition 2:* A **distant time query** is a spatio-temporal predictive query satisfying $t_q \geq t_c + d$, where $t_q$ and $t_c$ represent the query time and the current time respectively, and $d$ is a threshold of a distant time ($t_q < T$, $0 < d < T$).

Though the boundary between distant time and non-distant time is application-dependent, intuitively it can be the maximum prediction lengths which existing motion functions assume.

## IV. Trajectory Pattern Discovery

Our approach towards retrieving trajectory patterns is essentially similar to that of mining association rules since we bring the form of trajectory pattern from that of the rules. Therefore, the discovery process can be divided into two major components: (1) to detect frequent regions (i.e., corresponding to frequent items) and (2) to derive trajectory patterns (i.e. corresponding to association rules) from the frequent regions.

For the first component, we adopt the periodical pattern mining methods [10] which discover objects' patterns that follow the same routes over regular time intervals. The methods decompose the whole trajectory into $\lfloor \frac{n}{T} \rfloor$ sub-trajectories and group all the locations $G_t$ having the same time offset $t$ in each sub-trajectory. They then apply the density-based clustering algorithm DBSCAN [18] to find clusters (frequent regions) for each time offset $t$. In this case, $MinPts$ and $Eps$ parameters of DBSCAN play the same role as $support$ of mining frequent item sets.

For the second component, we modify the *apriori* algorithm [19] to generate trajectory patterns from the frequent regions discovered. The key idea of the apriori algorithm is to generate candidate item sets of length $k$ from candidate item sets of length $k-1$. In this process, we prune some candidate item sets which are unnecessary for prediction as follows:

- Since a trajectory pattern is a monotonically increasing sequence in terms of the time offset associated with each region, all patterns which contradict this constraint are pruned. It means that we do not predict past or current positions from future movements. For example, $R_3^1 \wedge R_2^1 \longrightarrow R_1^1$ is not a qualified pattern.
- Our prediction approach does not require any rules that have multiple items in their consequences. Suppose that there are two trajectory patterns $R_1^0 \wedge R_2^0 \xrightarrow{0.9} R_3^0$ and $R_1^0 \wedge R_2^0 \xrightarrow{0.8} R_3^0 \wedge R_4^0$. When a query is issued with $t_q = 3$, our query processor will always select the first pattern as it has a higher confidence (both have the same premises). According to the *apriori* algorithm, rules having more

than one item in the consequence are created by rules having one item in the consequence. This implies that $R_1^0 \wedge R_2^0 \longrightarrow R_3^0$ and $R_1^0 \wedge R_2^0 \longrightarrow R_4^0$ must exist if there is a pattern $R_1^0 \wedge R_2^0 \longrightarrow R_3^0 \wedge R_4^0$. By Theorem 1, we can eliminate $R_1^0 \wedge R_2^0 \longrightarrow R_3^0 \wedge R_4^0$.

*Theorem 1:* Suppose that we have two patterns $P_1$: $s_1 \xrightarrow{c_1} f_1$ and $P_2$: $s_1 \xrightarrow{c_1} f_1 \wedge s_2$, where $f_1$ denotes a frequent region, $s_1$ and $s_2$ denote sequences of frequent regions respectively. $P_2$ having multiple frequent regions in its consequence is never used for prediction.

**Proof:** Let $N(s_1)$ denote the number of $s_1$ occurred in all the trajectory patterns discovered, $N(s_1, f_1)$ denote the number of $s_1$ and $f_1$ occurred together, $N(s_1, f_1, s_2)$ denote the number of $s_1$, $f_1$ and $s_2$ occurred together. The confidence $c_1$ of pattern $P_1$ is calculated as $\frac{N(s_1,f_1)}{N(s_1)}$ and $c_2$ of pattern $P_2$ is computed by $\frac{N(s_1,f_1,s_2)}{N(s_1)}$. Because $N(s_1, f_1) \geq N(s_1, f_1, s_2)$, $c_1 \geq c_2$ is always true. Since a pattern with a higher confidence is always selected for prediction, $P_2$ is never used for prediction. □

Pruning rules reduces not only the number of trajectory patterns to be produced but also the computing time for the trajectory pattern mining. Finding rules with multiple items in their consequences is computationally expensive due to the use of recursive calls in the Apriori algorithm. With these constraints, the pattern generation skips many of the recursive calls, hence, the computing time is significantly reduced. According to our experiments, 58% of trajectory patterns were reduced by the pruning effect.

## V. Trajectory Pattern Tree

In this section, we present our indexing method called *Trajectory Pattern Tree* (TPT), which is a variant of *Signature tree* [20]. Signature tree is a dynamic balanced tree and specifically designed for signature bitmaps. Each node contains entries of the form <*sig, ptr*>. In a leaf node entry, *sig* is the signature of the transaction and *ptr* is a transaction id. Each internal node entry is the logical *OR* on all signatures in its subtree. Given a query transaction $Q$, *sig(Q)* traverses the signature tree in a depth-first fashion. If it returns zero from the bitwise operation *AND* between *sig(Q)* and *sig(entry)*, it means there is no matched transaction indexed in the subtree and the traversal stops. Otherwise *sig(Q)* continues to its subtree until reaching the leaf node which has common items with $Q$.

TPT has a similar tree structure to signature tree but different leaf nodes. Each leaf node contains entries of the form $< pk, c, p >$, where $pk$ is the pattern key of a trajectory pattern, $c$ is its corresponding confidence and $p$ is the region key pointer which represents the consequence of the pattern. Fig. 4 shows an example of TPT indexing the trajectory patterns in Table III.

The key difference between signature tree and TPT is how to encode signatures to build a tree. In TPT, we encode a trajectory pattern to a signature, called a *pattern key*. The pattern key is designed for efficient retrieval of the similar patterns to a given object's recent movements and a query time.

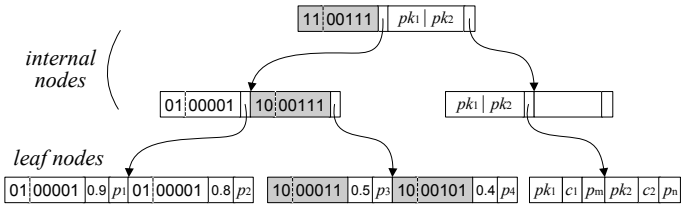



Fig. 4. An Example of TPT

### A. Pattern Key

A pattern key is the symbolization of a trajectory pattern. It is composed of two parts: premise key and consequence key. The premise key embodies the premise of a trajectory pattern and the consequence key represents the time offset of its corresponding consequence.

***Premise key*** : we sort all the frequent regions by the time offset associated with the regions. Unique region $id$s are given to each frequent region according to the order. We then encode *region keys* by a hash function $2^{id}$. Table I shows an example with 5 frequent regions presented in Fig. 3. Note that we do not need to store the actual region keys in the table because they can be obtained by the hash function. We leave them in the region key table for easy presentation. The length ($l_p$) of every region key is equal to the number of frequent regions.

| Frequent region | Region id | Region key |
|---|---|---|
| $R_0^0$ | 0 | 00001 |
| $R_1^0$ | 1 | 00010 |
| $R_1^1$ | 2 | 00100 |
| $R_2^0$ | 3 | 01000 |
| $R_2^1$ | 4 | 10000 |

TABLE I
An Example of Region Keys

The premise of a trajectory pattern may involve several frequent regions. Based on the region key table, a premise key is composed by bitwise operation *OR* on all the region keys of the frequent regions involved. For example, the premise key for $R_0^0 \wedge R_1^0$ ($R_0^0 \wedge R_1^1$) is 00011 (00101). Every '1' in a premise key represents an actual frequent region in the premise. It is worth noticing that we number the position of '1' in a premise key from right to left starting from 1. As a result, the following property applies, which will be used for premise similarity measures in Section VI-A:

*Property 1:* The '1' at position $i$ in the premise key is always closer or equals to the consequence time offset than the '1' at position $j$ if $i > j$.

***Consequence key*** : the construction of a consequence key is based on the trajectory pattern's consequence time offset $t$. We collect and sort all the time offsets associated with the consequences of trajectory patterns discovered. We then assign $id$s to each time offset with the same hash function used for the region keys. The length of the consequence key is equal

to the number of time offsets of consequences involved. Thus, it is always smaller than or equal to the length of a region key. All the time offsets and their $ids$ are maintained in the consequence key table. Table II provides an example having 2 time offsets involved in the consequences of the patterns in Fig. 3.

| Time offset | Time id | Consequence key |
|---|---|---|
| 1 | 0 | 01 |
| 2 | 1 | 10 |

TABLE II
AN EXAMPLE OF CONSEQUENCE KEYS

***Pattern key*** : we design a pattern key as the representation of a trajectory pattern. We encode the pattern key by combining the consequence key and the premise key of the trajectory pattern. Specifically, we place the consequence key first followed by the premise key. Table III shows pattern key examples for the trajectory patterns in Fig. 3. The trajectory patterns' consequence keys and premise keys are derived from Table I and Table II respectively. Note that a pattern key may represent more than one trajectory pattern (e.g., 0100001 in Table III). Since multiple frequent regions can exist at one consequence time offset, such frequent regions have the same consequence key. This may cause the case of having the same pattern key among different trajectory patterns.

| Trajectory pattern | Pattern key |
|---|---|
| $R_0^0 \xrightarrow{0.9} R_1^0$ | 0100001 |
| $R_0^0 \xrightarrow{0.8} R_1^1$ | 0100001 |
| $R_0^0 \wedge R_1^0 \xrightarrow{0.5} R_2^0$ | 1000011 |
| $R_0^0 \wedge R_1^1 \xrightarrow{0.4} R_2^1$ | 1000101 |

TABLE III
AN EXAMPLE OF PATTERN KEYS

***Pattern key operations*** : we define a set of operations that will be used for TPT construction and search. Let $\&$, $|$ and $\oplus$ denote the bitwise operations AND, OR, and XOR.

- **Union**($pk_1$, $pk_2$, ..., $pk_n$): given a set of pattern keys $pk_1$, $pk_2$, ..., $pk_n$, returns a new pattern key which is formed as $pk_1|pk_2|...|pk_n$.
- **Size**($pk$): given a pattern key $pk$, returns the number of '1's in $pk$.
- **Contain**($pk_1$, $pk_2$): given two pattern keys $pk_1$ and $pk_2$, returns true if $pk_1 \& pk_2 = pk_2$.
- **Difference**($pk_1$, $pk_2$): given two pattern keys $pk_1$ and $pk_2$, returns Size($pk_1 \oplus (pk_1 \& pk_2)$).
- **Intersect**($pk_1$, $pk_2$): let $ck_1(ck_2)$ and $rk_1(rk_2)$ denote the consequence key and the premise key of $pk_1(pk_2)$. If Size$(ck_1 \& ck_2) > 0$ and Size$(rk_1 \& rk_2) > 0$, returns true. Otherwise, returns false.

Notice that the **Intersect** operation checks if two given pattern keys have common '1's on both premise key and consequence key.

### B. Insertion

We assume the system deals with both static data (historical trajectory data) and dynamic data (newly incoming trajectory data). The system uses bulk loading to build TPT for the static data. When a certain amount of new data is accumulated, the system mines new patterns and adds them up to TPT by using the insertion algorithm.

The insertion algorithm follows a general insertion procedure of building a multi-dimensional balanced tree, such as R-tree. It first invokes *ChooseLeaf* algorithm (Algorithm 1) to find a node where a pattern key $pk$ should be inserted. If the node has free space, $pk$ is inserted into it, otherwise it splits the node. How to split is similar to that in signature tree and R-tree and thus it is not presented in this paper.

**Algorithm 1** ***ChooseLeaf***

**Input:**
the root $R$ of TPT, pattern key $pk$
**Output:**
a leaf node where $pk$ is placed
**Description:**
1: Set $N$ to be $R$ node;
2: **if** $N$ is a leaf **then**
3: return $N$;
4: **else**
5: **if Contain**($e_i$, $pk$), ($e_i$ is an entry's pattern key, $e_i \in N$) **then**
6: set $E$ to be the entry with the smallest **Size**($e_i$);
7: **else if Intersect**($e_i$, $pk$), ($e_i \in N$) **then**
8: set $E$ to be the entry with the smallest **Difference**($pk$, $e_i$). Resolve ties by set $E$ with the smallest **Size**($e_i$).
9: **else**
10: set $E$ to be the entry with the smallest **Difference**($pk$, $e_i$). Resolve ties by set $E$ with the smallest **Size**($e_i$).
11: set $N$ to be the node that is pointed by $ptr$ associated with $E$ and repeat from Step 2;

Now we describe some intuitions behind the *ChooseLeaf* algorithm. Let $pk$ denote a pattern key to be inserted. If the pattern key of an entry $e$ in an internal node can contains $pk$, we do not need to enlarge the space to place $pk$ in $e$. Likewise R-tree, we follow the pointer of the current entry. If there are several $e$s containing $pk$ (Line 5 and 6), the pattern key with the smallest size is chosen.

When there is no such pattern key of $e$ that can contain $pk$, we next look at if there is an intersection between $e$ and $pk$ on both consequence keys and premise keys (Line 7 and 8). This condition is useful for efficient query processing, which will be discussed in Section VI. This aspect cannot be achieved by the construction algorithm of signature tree. If neither **Contain** nor **Intersect** condition is satisfied, only **Difference** operation will be considered to find the best place to insert $pk$.

### C. Search

Given a query (i.e., the object's recent movements $m_q$ and query time $t_q$), we first encode its pattern key $q$. Specifically, we investigate which frequent regions the object has visited recently from $m_q$. Such a sequence of frequent regions is composed to a premise key by looking the *region key table*. Meanwhile, the query time $t'_q$ is transferred into its time offset $t_q = t'_q \bmod T$. Next, we generate a consequence key by finding $t_q$ in the *consequence key table*. Now $q$ is encoded

by combining the premise key and the consequence key as discussed in Section V-A.

After the query pattern key $q$ is obtained, TPT retrieves all the trajectory patterns satisfying the condition that **Intersect**($pk$, $q$) is true, where $pk$ is a pattern key indexed in TPT. We employ a depth-first search method. Let *e* denote an entry's pattern key in an internal node in TPT. If **Intersect**$(e, q)$ is true, there may exist a pattern key in its sub-tree that could match the query key. The search algorithm follows all such entries until it reaches the leaf node. If **Intersect**$(e, q)$ is false, we can safely stop searching the subtree pointed by this entry. For any leaf node reached, its entries that intersect with $q$ are returned.

| *Symbols* | *Descriptions* |
|---|---|
| $R_t^j$ | $j^{th}$ frequent region at time offset $t$ |
| $P, c$ | trajectory pattern, its confidence |
| $m_q$ | object's recent movements |
| $pk, q$ | pattern key, query pattern key |
| $rk, rkq$ | premise key, query premise key |
| $ck$ | consequence key |
| $S_p$ | similarity between pattern and query |
| $S_r, S_c$ | premise similarity, consequence similarity |
| $t, t_q$ | time offset, query time offset |
| $t_c, t_e$ | current time offset, time relaxation length |
| $d$ | threshold for distant time query |

TABLE IV
TABLE OF NOTATIONS

## VI. HYBRID PREDICTION ALGORITHM

In this section, we present our query processing method, *Hybrid Prediction Algorithm*, which takes advantages of an object's pattern information as well as its motion function. The motion function can be any type (e.g., a linear function) but Recursive Motion Function (RMF) [6] is used for this study since it has higher accuracy of prediction than others. For the query processing, two different techniques are adopted with respect to different query types.

For non-distant time queries, we use the *Forward Query Processing* which treats recent movements of an object as an important parameter to predict near future locations. A set of qualified candidates will be retrieved and ranked by their premise similarities to the given query. We then select top-$k$ ($k$ is given by user) patterns and return the centers of their consequences as answers. Recall that a consequence is a region. When there is no candidate, RMF will be called to answer the query.

For distant time queries, since recent movements become less important for prediction, the *Backward Query Processing* is used. Its main idea is to assign lower weights to premise similarity measure and higher weights to consequences which are closer to the query time in the ranking process of the pattern selection.

### A. Premise Similarity Measure

It is clear that more recent movements potentially have greater effect on future movements. Let us consider an example which Jane has a trajectory pattern *Home* $\rightarrow$ *City* $\rightarrow$ *Work place* ($R_0^0 \wedge R_1^0 \xrightarrow{0.5} R_2^0$). To predict the location at $t_q = 2$, $R_1^0$ (City) plays more important role compared with $R_0^0$ (Home) because the time offset 1 associated with $R_1^0$ is closer to the consequence time offset 2. Therefore, we put more weights to the frequent region in a premise key, which is closer to the current time.

Measuring the weighted similarity is performed efficiently by using TPT. Through Property 1, we can conclude that the '1' with a higher position in the premise key is more important than the '1' with a lower position. Therefore, we assign a weight to each '1' in the premise key based on the numbered position to represent its importance. Let $\omega_i$ denote the weight of '1' at position $i$ in the premise key $rk$. It can be calculated by any weight function such as a linear function $\omega_i = \frac{i}{\sum_{i=1}^{Size(rk)} i}$, quadratic $\omega_i = \frac{i^2}{\sum_{i=1}^{Size(rk)} i^2}$, exponential $\omega_i = \frac{2^i}{\sum_{i=1}^{Size(rk)} 2^i}$, and factorial $\omega_i = \frac{i!}{\sum_{i=1}^{Size(rk)} i!}$. By the linear function, for premise key 00011, the '1's at position 2 has a larger weight ($\frac{2}{3}$) than that of the '1' at position 1 ($\frac{1}{3}$). According to our experiments, the linear and the quadratic functions showed better prediction results among the weight functions.

Given an object's recent movements and a query time, we first generate the pattern key $q$, which was described in Section V-C. We then compare the premise key $rkq$ of $q$ with a premise key $rk$ of a trajectory pattern $pk$ in TPT. The premise similarity between $rk$ and $rkq$ is measured by summing up all the weights of '1's in $rk$ which are also in $rkq$. That is,

$$S_r = \sum_{i=1}^{Size(rk\&rkq)} \omega_i \quad (0 \leq S_r \leq 1) \tag{1}$$

For example, the premise similarity between $rk = 00011$ and $rkq = 00011$ is 1. It means that they are exactly the same. Likewise, the similarity between $rk = 00011$ and $rkq = 00010$ is $\frac{2}{3}$.

### B. Forward Query Processing (FQP)

FQP is designed to handle non-distant time queries that an object's recent movements are more important for location prediction. It retrieves all the trajectory patterns from TPT which satisfy two conditions: (1) the premise of the trajectory pattern is similar to that of the query pattern key, (2) its corresponding consequence time offset is the same as the query time. The qualified candidates are then ranked by their premise similarities and confidences. The center of each consequence of the top-$k$ trajectory patterns will be returned as the query answers.

Given a query pattern key, the search method of TPT can efficiently retrieve all the trajectory patterns which satisfy the above two conditions. The corresponding confidences and predictive locations are obtained from leaf nodes.

Intuitively, the larger the premise similarity and the larger the confidence, the greater the chance that an object will follow the pattern and move to its consequence. Since the premise similarity and the confidence are completely independent evidences, *Compound Probability* can be used to integrate

both. Therefore, we weight each qualified pattern with pattern key $pk$ by taking both *premise similarity* and *confidence* into consideration as follows:

$$S_p(pk, q) = S_r \times c \quad (0 \leq S_p \leq 1) \tag{2}$$

The pattern having a higher weight is ranked higher, i.e., higher probability that the query object is likely to follow.

Suppose an example that Jane's recent movements are $R_0^0$ and $R_1^0$, and $t_q = 2$. We get her query pattern key $q$ as 1000011 from Table I and Table II. All the entries satisfying **Intersect**$(e, q)$ (for internal nodes) or **Intersect** $(pk, q)$ (for leaf nodes) will be retrieved (shadow entries in Fig. 4). Therefore, there are two qualified candidates for this query. By Equation 2, if we apply the linear weight function, $S_p(1000011, 1000011) = 1 \times 0.5 = 0.5$ and $S_p(1000101, 1000011) = 0.33 \times 0.4 = 0.132$. Therefore, the consequence $R_2^0$ is the most probable region that Jane would move to at $t_q = 2$. If $k = 1$, only the center position of $R_2^0$ will be returned. Otherwise, both will be returned.

In some cases, we may not find any pattern that is similar to the object's recent movements. For the cases, we invoke a motion function to answer the query. The Forward Query Processing is shown in Algorithm 2.

**Algorithm 2 Forward Query Processing (FQP)**

**Input:**
query pattern key $q$, $k$
**Output:**
$k$ predicted locations
**Description:**
Get candidate trajectory patterns $C$ by searching TPT;
**if** $C \neq \emptyset$ **then**
  Rank the patterns in $C$ by Equation (2);
  Return the consequence centers of top $k$ patterns;
**else**
  Call motion function;

### C. Backward Query Processing (BQP)

For distant time queries, objects' recent movements may not be much assistance to predict future location. In this case, we try to find the locations where the object often visits at $t_q$ or near $t_q$. These locations are more reasonable than those predicted by the object's recent movements. For example, assume that the current time is 8:00 a.m. and the query time is 4:00 p.m.. The locations between 3:58 p.m. and 4:02 p.m. from trajectory patterns are expected to be more sensible than the prediction based on the movements around 8:00 a.m..

Therefore, compared with FQP which requires intersection constraints on both the premise key and the consequence key, BQP gives up the constraint for the premise key. Further more, the constraint for the consequence key is also relaxed using an additional parameter *time relaxation length* $t_\varepsilon$. Any trajectory pattern whose consequence time offset falls in the time interval $[t_q - t_\varepsilon, t_q + t_\varepsilon]$ is a qualified candidate, regardless its premise similarity.

We also weight the candidate's consequence key based on the time distance between its associated time offset $t$ and the query time $t_q$. Intuitively, a consequence with a closer $t$ to $t_q$ is more likely to be considered as a correct answer. Therefore, *consequence similarity* is defined as follows:

$$S_c = 1 - \frac{|t_q - t|}{t_\varepsilon + 1} \quad (0 \leq S_c \leq 1) \tag{3}$$

For distant queries, the pattern similarity is computed by taking consequence similarity into consideration as:

$$S_p(pk, q) = (S_r + S_c) \times c \tag{4}$$

where $S_r$ is calculated by Equation 1.

Let $t_c$ denote the current time offset. As $t_q$ is further from $t_c$, the recent movements become less important. Therefore, for distant time queries, the importance of premise similarity $S_r$ needs to be penalized. To reflect this idea, we further redefine $S_p$ as follows:

$$S_p(pk, q) = (S_r \times \frac{d}{t_q - t_c} + S_c) \times c \tag{5}$$

where $0 < \frac{d}{t_q - t_c} \leq 1$.

In some cases, there is no qualified pattern whose consequence time offset falls into the time interval $[t_q - t_\varepsilon, t_q + t_\varepsilon]$. BQP will incrementally enlarge the time interval until a pattern is found. Through our experiments, the best prediction accuracy regarding to the time relaxation length $t_e$ was observed when $1 \leq t_e \leq 3$. The Backward Query Processing is presented in Algorithm 3.

**Algorithm 3 Backward Query Processing (BQP)**

**Input:**
query pattern key $q$, time relaxation length $t_\varepsilon$, $k$
**Output:**
$k$ predicted locations
**Description:**
Set i = 1;
Get candidate trajectory patterns $C$ by searching TPT for patterns in interval $[t_q - i \times t_\varepsilon, t_q + i \times t_\varepsilon]$ ;
**if** $C \neq \emptyset$ **then**
  Rank the patterns in $C$ by Equation (5);
  Return the consequence centers of top $k$ patterns;
**else**
  Set i = i + 1;
  **if** $t_q - i \times t_\varepsilon > t_c$ **then**
    Go to Step 2;
  **else**
    Call motion function

## VII. Experiments

The experiments in this study are designed for three objectives. First, we compare the prediction accuracy of our method, Hybrid Prediction Model (HPM), with Recursive Motion Function (RMF) that is the most accurate motion function in the literature. Second, we investigate the changes of prediction accuracy with various parameters in pattern discovery process. Lastly, we compare the query processing costs between HPM and RMF. We conducted a set of experiments on our prototype which was implemented in the C++ language on a Windows XP operating system. We also acquired the source code of RMF. Both prediction methods were run on a dual processor Intel Pentium4 3.0 GHz system with a memory of 512MB.

Due to the lack of accumulated real datasets, we generated four synthetic datasets. First, we obtained four different objects' trajectories as follows:

- **Bike**: A bike's movements were measured by a GPS mounted bicycle. The data was recorded while the bike went from a small town to another town in Australia over an eight hour period.
- **Cow**: As a part of the virtual fencing project in Australia, 13 cattle having GPS-enabled ear-tags provided their location information. We used one cow's trajectory among them.
- **Car**: The movements of a private car for one and a half hours were measured by a GPS device while it was following the Tehran road in Seoul, Korea.
- **Airplane**: Some points were sampled from real data (road networks in California) to serve as airports, then random locations were synthetically generated on the segment connecting two random airports.

We then generated 199 similar trajectories having $T = 300$ to each original trajectory. Thus, each dataset had 200 sub-trajectories (e.g., a car's 200 days movements, and each day had 300 positions). The extent of each dataset was normalised to [0,10000] in both $x$ and $y$ axis. For the generation, we modified the periodic data generator [10] to be able to produce trajectories implying patterns. We set most parameters of the generator to the same values as the study except the probability $f$ that a generated trajectory was similar to the given trajectory. We set different probabilities to each data generation (Bike>Cow>Car>Airplane). Therefore, more trajectory patterns can be discovered in the Bike dataset while Airplane had week movement patterns.

### A. Prediction Accuracy Comparison

In the first set of experiments, we compare the prediction accuracy of HPM with that of RMF under varying conditions. A prediction error is measured as the distance between a predicted location and its actual location. We test 50 queries for both RMF and HPM and average their errors. We set HPM parameters as follows: the number of results returned $k = 1$, the number of sub-trajectories to discover trajectory patterns = 60, the threshold for distant time query $d = 60$, the maximum distance to neighbor points DBSCAN $Eps = 30$, the minimum number of neighborhood points in $Eps$ $MinPts = 4$, and $minimum\ confidence = 0.3$ for all datasets. RMF parameters are set for the best performance in terms of accuracy based on its experimental discussions.

As expected, our method shows very low errors regardless of prediction length while the errors of RMF rise significantly as the prediction length increases in Fig. 5. Especially in the Car dataset, this observation is prominent because it has many sudden changes of direction on road intersections. These results prove that using HPM for a distant time query is obviously more precise and sensible. Another observation found in the Airplane dataset demonstrates that HPM's accuracy is not as good as that in other datasets. The reason is because the dataset does not contain strong trajectory patterns. Thus the prediction of HPM is not supported by enough pattern information.

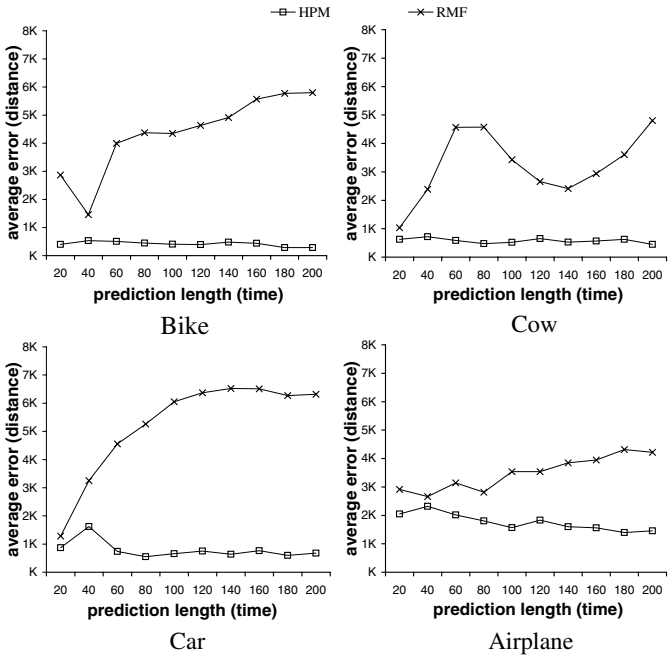



Fig. 5. Effect of Prediction Length

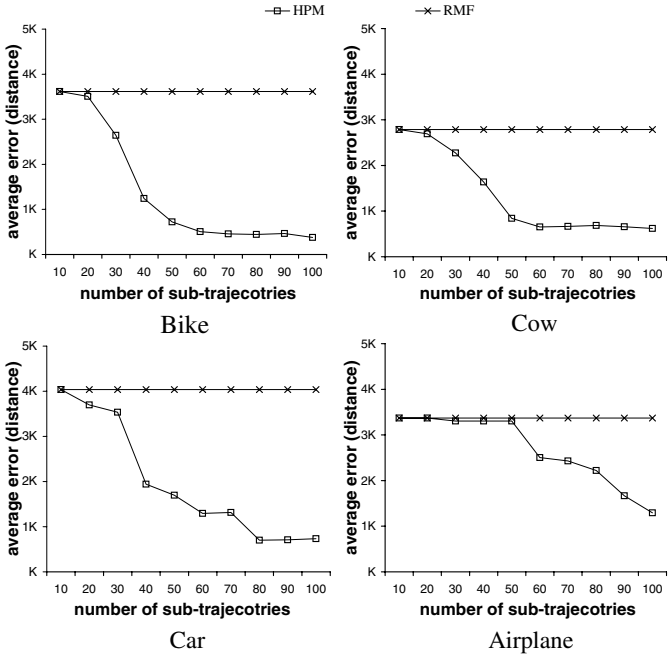



Fig. 6. Effect of Sub-trajectories

With a large amount of sub-trajectories, we can find not only a greater number of patterns but also more accurate patterns. This idea is clearly shown in Fig. 6 (prediction length=50). HPM's errors remain almost constant and close to those of RMF for the first few tens of sub-trajectories, yet, they show a steep decrease after this period for all the datasets. This fact

reflects that HPM can become dramatically more precise when a proper amount of sub-trajectories have been accumulated for the trajectory pattern discovery. Another important view of the figure is that HPM errors do not exceed RMF errors throughout the experiments.

### B. Effect of Discovery Parameters

We next study the effect of $Eps$ which is closely related to the number of frequent regions. When $Eps$ is large, it may include many points within the $Eps$ range, hence a cluster (frequent region) can be easily constructed. Eventually, it can generate many trajectory patterns and potentially achieve good prediction precision.

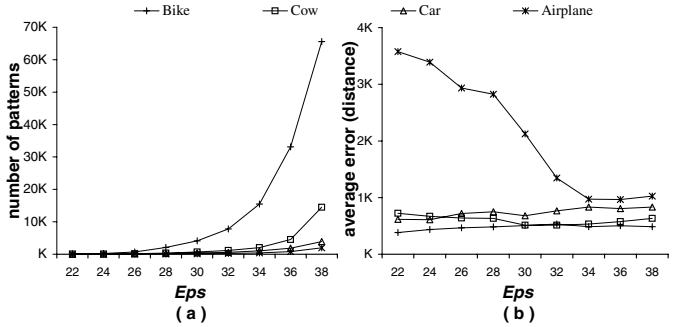



Fig. 7. Effect of $Eps$

As seen in Fig. 7(a), the number of trajectory patterns increases dramatically as the value of $Eps$ grows. One interesting observation is that if sufficient patterns are extracted, the exceeded number of patterns seldom affects prediction accuracy. This characteristic is obvious in the Bike dataset. Although the number of patterns grows up to 65,558 as $Eps$ increases, the accuracy remains almost the same (Fig. 7(b)) regardless of the increasing number of patterns (Fig. 7(a)). On the other hand, the growth of $Eps$ greatly affects the prediction of the Airplane dataset since its number of patterns is not sufficient until $Eps = 34$.

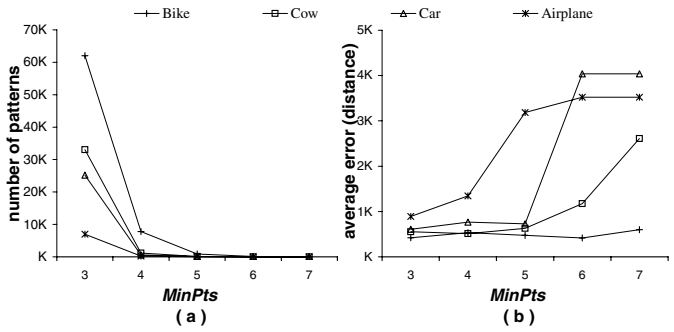



Fig. 8. Effect of $MinPts$

We also investigate the effect of $MinPts$. In general, a cluster (frequent region) needs $MinPts$ number of points to be built. Therefore, a high value of $MinPts$ may cause a small number of trajectory patterns. As a result, prediction based on trajectory patterns could be affected by $MinPts$. In Fig. 8(a), the number of trajectory patterns is considerably reduced as the number of $MinPts$ increases. Due to the small number of trajectory patterns, the prediction errors rise significantly (Fig. 8(b)).

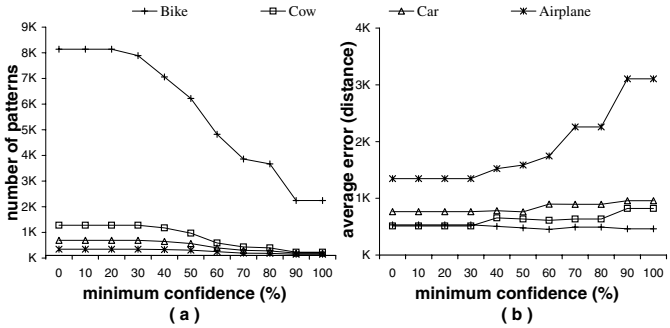



Fig. 9. Effect of $minimum\ confidence$

The effect of $minimum\ confidence$ for prediction is studied too. For the Bike dataset, significant reduction of its number of patterns is observed in Fig. 9(a) while its accuracy shows slight changes in Fig. 9(b). It implies that only certain numbers of patterns are useful for prediction though many patterns are discovered. In contrast, for the Airplane dataset that has the least number of patterns, the change of confidence value affects the prediction accuracy greatly when the confidence value reaches 60%, i.e., the number of patterns for the Airplane dataset becomes insufficient as shown in Fig. 9(a).

The values of discovery parameters are greatly affected by data distribution and thus highly application-dependant. $MinPts$ and $Eps$ play the same role as $support$ in association rule mining, and thus setting them implies how the system decides the 'frequency' of an object's appearing. According to our experimental experiences, we recommend the system to find many frequent regions by setting relatively large $Eps$ and low $MinPts$. Unnecessary frequent regions for prediction will be pruned by setting $minimum\ confidence$ at next step. If an application predicts locations at far distant time, $minimum\ confidence$ can be determined to a low value because pattern information is more reliable than using motion functions in the case.

### C. Query Cost Comparison

In these experiment sets, we compare the efficiency of HPM with RMF in terms of query response time. The results were averaged by 30 queries to each dataset. From Fig. 10, the query cost of HPM decreases considerably as the number of discovered patterns increases. This is caused by a less number of RMF calls from HPM since it is more likely for HPM to find available patterns for prediction when more patterns are discovered. Note that RMF involves high computational cost ($n^3$ due to Single Value Decomposition, where $n$ is the number of historical timestamps used) to construct and train itself while HPM only searches for similar patterns indexed by TPT. As more patterns are available, it is less likely for HPM to call RMF for prediction. This experiment shows that HPM is more efficient than RMF.

Lastly, we study the performance of TPT. Fig. 11(a) demonstrates storage requirements of TPT. The length of a pattern

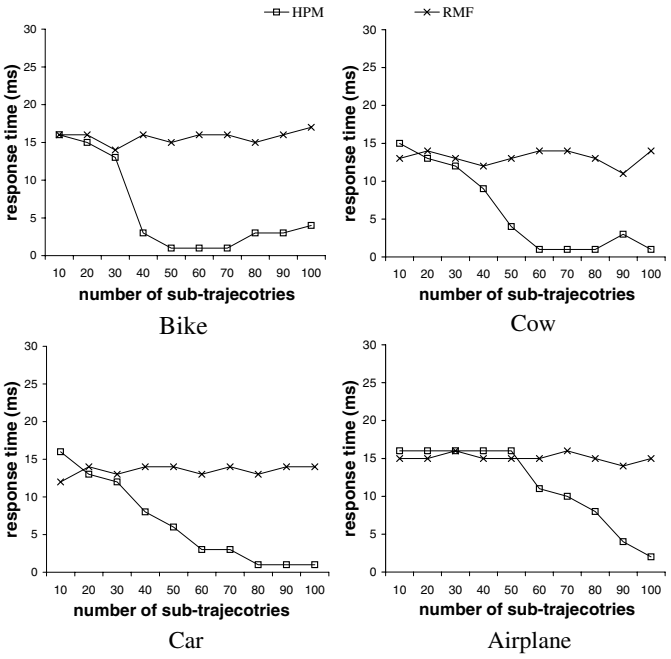



Fig. 10. Query Response Time

key in an entry is directly related to the number of frequent regions (see SectionV-A), thus, TPT with the bigger number of frequent regions is more subjective to the storage size as the number of patterns increases. According to our experiments, however, the storage size for the largest pattern information of our experiments is still reasonably small. In spite of the clear variance of storage size, query response times of TPT remain almost constant while those of the brute-force method increase tremendously as the number of patterns grows (Fig. 11(b)).

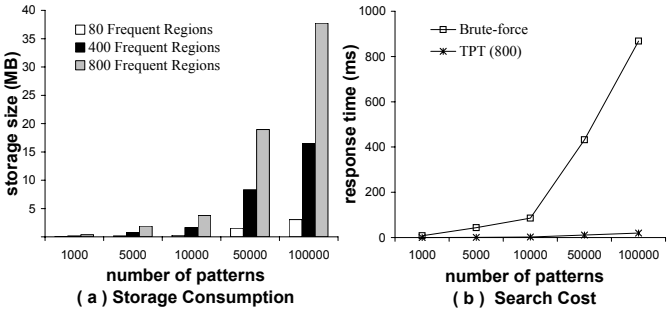



Fig. 11. Performance of TPT

## VIII. Conclusion

In this paper, we presented a novel approach which forecasted an object's future locations in a hybrid manner utilizing not only motion function but also objects' movement patterns. Specifically, trajectory patterns of objects were defined and discovered in order to evaluate spatio-temporal predictive queries. We then introduced the trajectory pattern tree indexing the patterns discovered for efficient query processing. We also proposed the hybrid prediction algorithm which could provide accurate results for both distant time queries and non-distant time queries. Our comprehensive experiments demonstrated that our techniques were more accurate and efficient than the existing predictive methods.

## Acknowledgment

The work reported in this paper is supported by grants DP0773483 and DP0663272 from the Australian Research Council. National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

## References

[1] Y. Tao and D. Papadias, "Spatial queries in dynamic environments," *TODS*, vol. 28, no. 2, pp. 101–139, 2003.
[2] J. M. Patel, Y. Chen, and V. P. Chakka, "Stripes: an efficient index for predicted trajectories," in *SIGMOD*, 2004, pp. 635–646.
[3] C. S. Jensen, D. Lin, and B. C. Ooi, "Query and update efficient b+-tree based indexing of moving objects." in *VLDB*, 2004, pp. 768–779.
[4] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, "Indexing the positions of continuously moving objects," in *SIGMOD*, 2000, pp. 331–342.
[5] Y. Tao, D. Papadias, and J. Sun, "The tpr*-tree: An optimized spatio-temporal access method for predictive queries." in *VLDB*, 2003, pp. 790–801.
[6] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu, "Prediction and indexing of moving objects with unknown motion patterns," in *SIGMOD*, 2004, pp. 611–622.
[7] G. Yavas, D. Katsaros, O. Ulusoy, and Y. Manolopoulos, "A data mining approach for location prediction in mobile environments," *Data & Knowledge Engineering*, vol. 54, no. 2, pp. 121–146, 2005.
[8] Y. Ishikawa, Y. Tsukamoto, and H. Kitagawa, "Extracting mobility statistics from indexed spatio-temporal datasets." in *STDBM*, 2004, pp. 9–16.
[9] J. Yang and M. Hu, "Trajpattern: Mining sequential patterns from imprecise trajectories of mobile objects." in *EDBT*, 2006, pp. 664–681.
[10] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung, "Mining, indexing, and querying historical spatiotemporal data," in *SIGKDD*, 2004, pp. 236–245.
[11] G. Kollios, D. Papadopoulos, D. Gunopulos, and J. Tsotras, "Indexing mobile objects using dual transformations," *The VLDB Journal*, vol. 14, no. 2, pp. 238–256, 2005.
[12] C. C. Aggarwal and D. Agrawal, "On nearest neighbor indexing of nonlinear trajectories," in *PODS*, 2003, pp. 252–259.
[13] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, pp. 257–286, 1989.
[14] A. Bhattacharya and S. K. Das, "Lezi-update: an information-theoretic approach to track mobile users in pcs networks," in *MobiCom*, 1999, pp. 1–12.
[15] F. Verhein and S. Chawla, "Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases," in *DASFAA*, 2006.
[16] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias, "Spatio-temporal aggregation using sketches," in *ICDE*, 2004, p. 214.
[17] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, "Trajectory pattern mining," in *SIGKDD*, 2007, pp. 330–339.
[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *SIGKDD*, 1996, pp. 226–231.
[19] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *VLDB*, 1994, pp. 487–499.
[20] N. Mamoulis, D. W. Cheung, and W. Lian, "Similarity search in sets and categorical data using the signature tree." in *ICDE*, 2003, pp. 75–86.