Information Sciences 179 (2009) 2218–2231

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Mining frequent trajectory patterns in spatial–temporal databases

Anthony J.T. Lee *, Yi-An Chen, Weng-Chong Ip

Department of Information Management, National Taiwan University, No. 1, Section 4, Roosevelt Road, Taipei 10617, Taiwan, ROC

ARTICLE INFO

Article history:
Received 10 October 2007
Received in revised form 27 February 2009
Accepted 27 February 2009

Keywords:
Data mining
Frequent trajectory pattern
Location-based service
Spatial–temporal pattern
Spatial–temporal database

ABSTRACT

In this paper, we propose an efficient graph-based mining (GBM) algorithm for mining the frequent trajectory patterns in a spatial–temporal database. The proposed method comprises two phases. First, we scan the database once to generate a mapping graph and trajectory information lists (TI-lists). Then, we traverse the mapping graph in a depth-first search manner to mine all frequent trajectory patterns in the database. By using the mapping graph and TI-lists, the GBM algorithm can localize support counting and pattern extension in a small number of TI-lists. Moreover, it utilizes the adjacency property to reduce the search space. Therefore, our proposed method can efficiently mine the frequent trajectory patterns in the database. The experimental results show that it outperforms the Apriori-based and PrefixSpan-based methods by more than one order of magnitude.

1. Introduction

With advances in tracking technologies and the rapid improvement in location-based services, a large amount of spatial–temporal data has been collected in databases [8]. As a result, mining implicit and useful patterns in such databases has attracted increasing attention recently. The entities in a database that change their locations over time are defined as "moving objects", where the movement (i.e., the trajectory) of an object can be described as a sequence of spatial coordinates, each of which is associated with a time stamp.

Finding the frequently repeated trajectory patterns can help us analyze and predict the movements of objects. Temporal information in the patterns makes many applications that are only concerned with the spatial information more realistic and practical. For example, using temporal information to understand customers' movement (trajectory) patterns can help us make an appropriate recommendation or a mobile advertisement for them. In ecology, analyzing animals' movement routes can help us better understand their behavior or detect a strange phenomenon or disaster, especially if some animals' movement patterns change abruptly.

Therefore, in this paper, we propose an efficient algorithm that takes into account both spatial and temporal attributes to mine the frequent trajectory patterns. The proposed method comprises two phases. First, we scan the database once to generate a mapping graph and trajectory information lists (TI-lists). Then, we apply the proposed graph-based mining (GBM) algorithm to traverse the mapping graph and mine the frequent trajectory patterns in a depth-first search manner. Since our proposed algorithm uses the adjacency property to reduce the search space, we only need to extend a trajectory to the adjacent neighborhoods of the last location of the trajectory pattern. By using the mapping graph and TI-lists, the GBM algorithm can localize support counting and pattern extension to a small number of TI-lists. Thus, it is more efficient than the Apriori-based and PrefixSpan-based algorithms.

* Corresponding author. Tel.: +886 2 33661188; fax: +886 2 33661199.
E-mail addresses: jtlee@ntu.edu.tw (A.J.T. Lee), cookiemouse@gmail.com (Y.-A. Chen), r94017@im.ntu.edu.tw (W.-C. Ip).

0020-0255/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2009.02.016

The main contributions of this study are summarized as follows: (1) We propose a spatial–temporal pattern to describe a trajectory, where both the spatial and temporal attributes are simultaneously considered. (2) By using the mapping graph, the GBM algorithm can localize support counting and pattern extension to a small number of TI-lists. (3) The GBM algorithm exploits the adjacency property to mine frequent trajectory patterns and reduce the search space. (4) We propose an efficient mining algorithm for discovering frequent trajectory patterns without candidate generation.

The remainder of this paper is organized as follows: Section 2 discusses the related work. Section 3 describes the preliminary concepts and the problem definitions used throughout the paper. Section 4 presents the proposed algorithm in detail. Section 5 illustrates the performance of our approach. Finally, Section 6 describes concluding remarks and future work.

2. Related work

Mining frequent itemsets is one of the fundamental problems in the area of data mining. An itemset is considered frequent if its support is not less than a user-specified minimum support threshold, where the support of an itemset is defined as the percentage of transactions in the database that contain the itemset. Many itemset mining algorithms [1,15,17,24,36,38,39] have been proposed for mining the frequent itemsets in a database. Agrawal et al. [1] proposed an Apriori approach to mine frequent itemsets from transactional databases. Wang et al. [36] developed an approach to generate online association rules based on multi-dimensional pattern relations. An association rule is of the form $X \Rightarrow Y$, where $X \cup Y$ are frequent itemsets, the intersection of X and Y is an empty set, and the confidence of the rule is not less than a user-specified minimum confidence threshold. The confidence of the rule $X \Rightarrow Y$ is the percentage of transactions in the database containing X that also contain Y. Lee et al. [24] developed an Aprori-based approach to mine association rules from image databases. Yu et al. [38] presented a false negative approach to mining frequent itemsets from high speed transactional data streams. Yun [39] proposed an approach to mine weighted frequent itemsets.

Moreover, several sequential pattern mining algorithms [2,11,16,26,27,32,33,41] have been proposed. Srikant and Agrawal [33] proposed an Apriori-based approach, called AprioriAll, to generate sequential patterns level by level. Pei et al. [32] presented a method, called PrefixSpan, to reduce the search space and accelerate the mining process by using the projected databases. Ayres et al. [2] used a vertical bitmap representation to count the supports and mine sequential patterns from sequence databases. Giannotti et al. [11] demonstrated an approach to mine sequential patterns with temporal annotations. Zaki [41] developed an approach, called SPADE, to decompose the original mining problem into several smaller sub-problems so that each sub-problem can be independently solved in main memory. Leleu et al. [27] proposed a method, called Go-SPADE, which extends SPADE [41] to efficiently mine sequences containing consecutive repetitions of items.

The algorithms discussed above are non-constraint-based algorithms. The patterns obtained from those algorithms could be overwhelmingly numerous and probably hard to interpret. In contrast, the constraint-based algorithms incorporate the constraints to reduce the search space and mine the useful patterns for a certain purpose. Srikant et al. [34] considered the problem of integrating constraints into mining algorithms, where the constraints were Boolean expressions over the presence or absence of items into the association rule discovery algorithm. Ng et al. [28] introduced two classes of constraints: anti-monotonicity and succinctness, and proposed a mining algorithm for handling these classes of constraints within the Apriori framework. Grahne et al. [13] introduced the third class of constraint, monotonicity, to mine correlated patterns. Pei et al. [30,31] discussed the convertible constraints and proposed the method to push these constraints deep inside the FP-growth algorithm for frequent itemsets. Lee et al. [25] presented an approach to mine association rules with the multiple constraints constructed by multi-dimensional attributes. Chen and Hu [7] utilized the anti-monotonic property to confine the time spans within sequential patterns. Orlando et al. [29] used the gap constraint between events to facilitate marketing analysis. Gade et al. [12] discussed how to push aggregate constraints into mining processes. Garofalakis et al. [10] proposed an approach, called SPIRIT, to mine sequential patterns with regular expression constraints. Yun [40] used the weight constraints to efficiently mine sequential patterns in large sequence databases.

Furthermore, many graph mining algorithms [14,18,19,21–23,37] have been proposed. AGM [21] and FSG [23] use an Apriori-based approach to combine frequent subgraphs mined at the previous level to generate all candidates at the next level. Gudes et al. [14] takes edge-disjoint paths as basic graph elements to perform candidate generation. gSpan [37] generates the frequent subgraphs without candidate generation in a depth-first search manner. FFSM [18] exploits an algebraic graph framework called canonical adjacency matrix (CAM) tree to perform join-extension operations to unambiguously enumerate all frequent subgraphs and avoid subgraph isomorphism testing by maintaining an embedding set for each frequent subgraph. Jin et al. [22] presented a method based on the topological minor concept to eliminate some impossible vertices. Thus, the algorithm can reduce the search space and increase the mining efficiency.

However, the sequential pattern, itemset, and graph mining methods are not suitable for mining frequent trajectory patterns. Itemset and sequential pattern mining algorithms do not consider the spatial attribute, while the graph mining algorithms do not consider the temporal attribute. In the problem of mining trajectory patterns, the spatial and temporal attributes need to be considered simultaneously.

Recently, some trajectory mining methods that take spatial attributes into account have been proposed. Tsoukatos and Gunopulos [35] proposed an Aprioi-based method for mining sequences of spatial regions that appears periodically. Hwang et al. [20] presented an algorithm for mining a group of moving objects with the same movement patterns. Cao et al. [4–6] proposed a series of trajectory mining algorithms. In [4], the original data points are first transformed into a sequence of

Table 1
An example database D.

TID	Trajectory
T_1	(1,1,1), (1,2,2), (2,2,3), (2,3,5), (3,4,6), (3,5,8)
T_2	(1,3,2), (1,2,4), (2,2,5), (2,3,7), (3,4,8), (3,5,10)
T_3	(1,3,3), (1,2,4), (2,2,7), (2,3,10), (3,4,13), (2,5,15), (1,5,16)
T_4	(2,1,1), (2,2,4), (2,3,7), (3,4,10), (2,5,12)
T_5	(1,1,5), (1,2,6), (2,2,7), (2,3,9), (3,4,10), (3,5,12)
T_6	(2,1,4), (2,2,5), (2,3,8), (3,4,11), (2,5,13), (1,5,14)

segments. Then, the frequent segments are identified and frequent patterns are mined using an improved Apriori-based technique. In [6], the proposed method cuts a very long spatial–temporal sequence into many short periodic sequences and uses an approach similar to that in [4] to mine the frequent periodic patterns. Cao et al. [5] developed a method to find similar trajectories between different objects. Chung et al. [9] proposed an Apriori-based method to mine movement patterns, where the moving objects are transformed into a grid system and the frequent patterns are generated level by level. However, the trajectory patterns mined in [4–6,9,20,35] do not consider the time span between two adjacent locations. The patterns that do not capture the temporal information may lose some important information.

3. Preliminaries and problem definitions

A *reference space* of size $e \times e$ is a 2-dimensional space, where the reference space is a square, the length and width of the reference space are equal to e, e is an integer, and $e \geqq 1$. The x- and y-coordinates of a point g_i in the reference space are denoted by $g_i \cdot x$ and $g_i \cdot y$, respectively, where $g_i \cdot x$ and $g_i \cdot y$ are integers, $1 \leqq g_i \cdot x \leqq e$, and $1 \leqq g_i \cdot y \leqq e$. Consider a spatial–temporal database $D = \{T_1, T_2, \ldots, T_u\}$, where T_i is the trajectory of a moving object, $T_i = \langle (x_1,y_1,t_1),(x_2,y_2,t_2),\ldots,(x_m,y_m,t_m)\rangle$, (x_j,y_j) is the location of the moving object at time stamp t_j, x_j and y_j are integers, $1 \leqq i \leqq u$, and $1 \leqq j \leqq m$.

Definition 1. Given two points, g_1 and g_2, in the reference space, if $|g_1 \cdot x - g_2 \cdot x| \leqq 1$ and $|g_1 \cdot y - g_2 \cdot y| \leqq 1$, we say that g_1 is adjacent to g_2. Note that a point in the reference space has at most eight adjacent points.

Definition 2. The time span from one location (x_1,y_1,t_1) to an adjacent location (x_2,y_2,t_2) for a moving object is defined as $t_2 - t_1$.

Definition 3. A trajectory pattern (pattern for short) is defined as $\langle g_1^{d1} g_2^{d2} \cdots g_{k-1}^{dk-1} g_k \rangle$, where $g_i$is a point, $1 \leqq i \leqq k$, $k \geqq 2$, any two consecutive points are adjacent, d_j is the time span between two consecutive points, $d_j \leqq \theta$, $1 \leqq j \leqq k-1$, and θ is a user-specified maximum time span threshold. If $k = 1$, the pattern is denoted as $\langle g_1 \rangle$. A pattern with k points is called a k-pattern.

Definition 4. A trajectory $\langle (x_1,y_1,t_1),(x_2,y_2,t_2),\ldots,(x_m,y_m,t_m)\rangle$ contains a pattern $\langle g_1^{d1} g_2^{d2} \cdots g_{n-1}^{dn-1} g_n \rangle$, if there exists i such that $g_1 = (x_i,y_i), g_2 = (x_{i+1},y_{i+1}),\ldots,g_n = (x_{i+n-1},y_{i+n-1})$ and $d_j = t_{i+j} - t_{i+j-1}$, $1 \leqq i \leqq m-n$, and $1 \leqq j \leqq n-1$.

For example, $\langle (1,1,1),(1,2,2),(2,2,3),(2,3,5),(3,4,6),(3,5,8)\rangle$ contains $\langle (2,2)^2(2,3)^1(3,4)^2(3,5)\rangle$.

Definition 5. The *support* of a trajectory pattern is defined as the percentage of trajectories in the database that contain the pattern.

Definition 6. A trajectory pattern is said to be *frequent* if its *support* is not less than a user-specified minimum support threshold.

To mine frequent trajectory patterns, we exploit the adjacency property. That is, any two consecutive points in a pattern are adjacent. Let us consider Example 1. Since $\langle (2,2)^3(2,3)^3(3,4)^2(2,5)\rangle$ is contained by T_3, T_4, and T_6, its support is 3/6 = 50%. Thus, it is a frequent pattern. On the other hand, since only T_3 and T_6 contain $\langle (2,2)^3(2,3)^3(3,4)^2(2,5)^1(1,5)\rangle$, its support is 2/6 = 33.33% which is less than 50%. Thus, it is not a frequent pattern.

Example 1. Assume that the minimum support threshold is 50% and the maximum time span threshold is 3. A spatial–temporal database D containing six trajectories is shown in Table 1, where the reference space is of size 5×5, and each trajectory in the database is identified by its trajectory ID.

The objective of mining frequent patterns is to find all frequent trajectory patterns in a database with respect to the user-specified minimum support and maximum time span thresholds.

4. The proposed method

In this section, we discuss the proposed GBM (graph-based mining) algorithm for mining the frequent patterns. As mentioned earlier, the proposed algorithm comprises two phases. First, we transform all trajectories in the database into a mapping graph. For each vertex in the mapping graph, we use a data structure, called a Trajectory Information list (TI-list), to record all trajectories that pass through the vertex. Then, we apply the proposed mining algorithm to find all frequent patterns. We introduce the TI-list in Section 4.1, and then describe the GBM algorithm in detail in Section 4.2.

4.1. Trajectory information list

Let us consider a reference space of size $e \times e$. We map each point to a vertex and add an edge for each pair of adjacent points. Thus, we can obtain a mapping graph that represents the structure of the reference space.

Next, we construct a TI-list for each vertex in the mapping graph. The TI-list of a vertex stores all trajectories that pass through the vertex, and each element of the TI-list contains the trajectory ID and the time stamp. The number of elements in a TI-list is denoted as *TI.count*. To construct TI-lists, we first create the TI-lists that only have headers, and then scan the database. For each location on a trajectory, we add the trajectory's ID and time stamp to the corresponding TI-list. After scanning the database, if the support of a vertex is less than the minimum support threshold, the corresponding TI-list of the vertex is deleted.

The construction algorithm is shown in detail in Fig. 1. In steps 1–6, we create a TI-list with a header for each vertex. In steps 7–11, the trajectory ID and time stamp of each trajectory are added to the corresponding TI-list. In steps 12–16, if the support of a vertex is less than the minimum support threshold, its corresponding TI-list is deleted. Note that the corresponding vertices of the remaining TI-lists are frequent 1-patterns.

Let us consider the same situation depicted in Example 1. The mapping graph of the example database is shown in Fig. 2a. First, we create 25 TI-lists with headers. Then, we scan the database, add the trajectory ID and time stamp to the corresponding TI-list, and count the support for each vertex. The supports of six TI-lists are not less than the minimum support threshold as shown in Fig. 2b, where the trajectories passing through vertex (1,2) are recorded in the first TI-list.

Theorem 1. *The time complexity of the construction algorithm is bounded by* $O(e^2 + \sum_{i=1}^{u} PL_i)$*, where* PL_i *is the length of trajectory* T_i*, the length and width of the reference space are equal to e, and u is the number of trajectories in the database.*

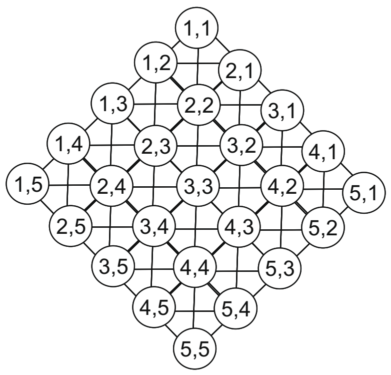
4.2. Mining frequent trajectory patterns

The GBM algorithm traverses the mapping graph in a depth-first search (DFS) manner to mine all frequent patterns. The algorithm is shown in Fig. 3 and contains a procedure called traverse, which is shown in Fig. 4. In the first step of the algorithm, we set the initial values of the variables, where *allST* records all frequent patterns mined and *currentST* records the frequent pattern being mined currently. Since each vertex in the mapping graph is a frequent 1-pattern, we extend it in steps 2–6. For each vertex in the mapping graph, we perform the following steps: (1) add the vertex to *currentST*; (2) add all trajectories containing *currentST* to *currentTI*, which is used to extend the pattern in the next level; and (3) call the traverse procedure to extend the *currentST*.

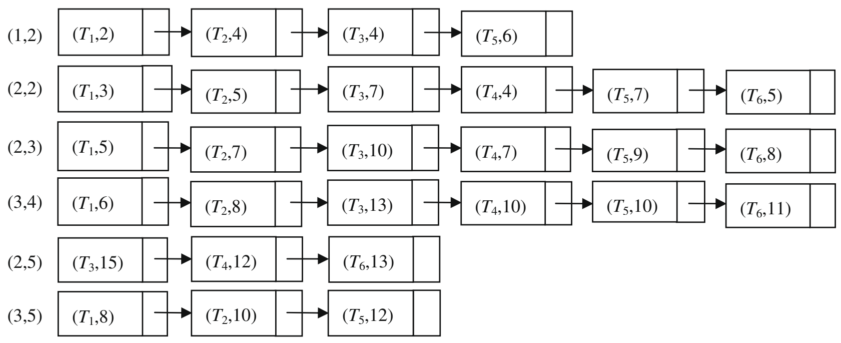
Definition 7. A *frequent neighbor* of a frequent pattern is a frequent vertex adjacent to the last vertex of the frequent pattern.

```
Algorithm: construction.
Input:    D: a spatial-temporal database, e: the length of the reference space, σ:
          user-specified minimum support threshold.
Output:   the mapping graph G with TI-Lists.
(1)       for i = 1 to e do
(2)           for j = 1 to e do
(3)               Create TI-list of vertex (i,j);
(4)               Add a header in the TI-list of vertex (i,j);
(5)           endfor;
(6)       endfor;
(7)       for each trajectory T_i in D do
(8)           for each time stamp t of T_i do
(9)               Add trajectory T_i and t to the TI-list of the corresponding vertex;
(10)          endfor;
(11)      endfor;
(12)      for each TI-list do
(13)          if  TI.count < σ*|D| then
(14)              Delete the TI-list;
(15)          endif;
(16)      endfor;
```

Fig. 1. The construction algorithm.



(a) A mapping graph.


(1,2) (T1,2) (T2,4) (T3,4) (T5,6)
(2,2) (T1,3) (T2,5) (T3,7) (T4,4) (T5,7) (T6,5)
(2,3) (T1,5) (T2,7) (T3,10) (T4,7) (T5,9) (T6,8)
(3,4) (T1,6) (T2,8) (T3,13) (T4,10) (T5,10) (T6,11)
(2,5) (T3,15) (T4,12) (T6,13)
(3,5) (T1,8) (T2,10) (T5,12)

(b) Six TI-lists in the mapping graph.

Fig. 2. A mapping graph and TI-lists.

```
Algorithm: GBM
Input:  G: mapping graph, σ: user-specified minimum support threshold, θ: maximum
        time span
Output: allST: all frequent patterns
(1) Set allST and currentST to ∅;
(2) for each vertex v in G do
(3)         Add v to currentST;
(4)         Add the trajectories containing currentST to currentTI;
(5)         allST =traverse (currentST, currentTI, σ, θ, allST);
(6) endfor;
(7) Return allST;
```

Fig. 3. The GBM algorithm.

Let us consider the database shown in Table 1. The frequent neighbors of the frequent pattern $\langle(1,2)^1(2,2)^2(2,3)^1(3,4)\rangle$ are vertices $(3,5)$, $(2,3)$ and $(2,5)$ because these vertices are frequent and adjacent to vertex $(3,4)$. Note that since a vertex has at most eight adjacent vertices, the number of frequent neighbors must be not greater than 8.

The input parameters of the traverse procedure are *currentST*, *currentTI*, σ, θ, and *allST*, where *currentST* is the frequent pattern being mined currently, *currentTI* is a list of trajectories containing *currentST*, σ is the user-specified minimum support threshold, θ is the maximum time span threshold, and *allST* records all the frequent patterns mined so far. The traverse procedure traverses the mapping graph in a DFS manner. It checks whether a vertex can be added to extend *currentST* and whether the number of trajectories containing the extended pattern is not less than the minimum support threshold. If this is the case, the extended pattern is frequent and added to *allST*. The traverse procedure recursively calls itself to extend *currentST*. The procedure will stop when no more vertices can be added to extend the *currentST*.

```
Procedure: traverse
Input:   currentST: the frequent pattern, currentTI: all trajectories containing currentST, $\sigma$:
         user-specified minimum support threshold, $\theta$: maximum time span threshold, allST:
         the set of all frequent patterns mined.
Output:  allST
(1)  Let s be the last vertex of currentST;
(2)    for k = 1 to neighbors do // neighbors is the number of frequent neighbors of currentST
(3)      Let $a_k$ be the $k^{th}$ frequent neighbors of currentST;
(4)      newTI=$\varnothing$;
(5)      for l =1 to $\theta$ do
(6)        for each trajectory t that appears in both currentTI and $a_k$'s TI-list do
(7)          if the time span between $a_k$ and s is equal to l then
(8)            Insert t into newTI;
(9)          endif;
(10)       endfor;
(11)       if the number of trajectories in newTI $\geq \sigma * |D|$ then
(12)         Add $a_k$ to currentST, where the time span between $a_k$ and s is equal to l;
(13)         Add currentST to allST;
(14)         allST=traverse(currentST, currentTI, $\sigma$, $\theta$, allST);
(15)       endif;
(16)     endfor;
(17)   endfor;
(18) Delete s from currentST;
(19) Return allST;
```

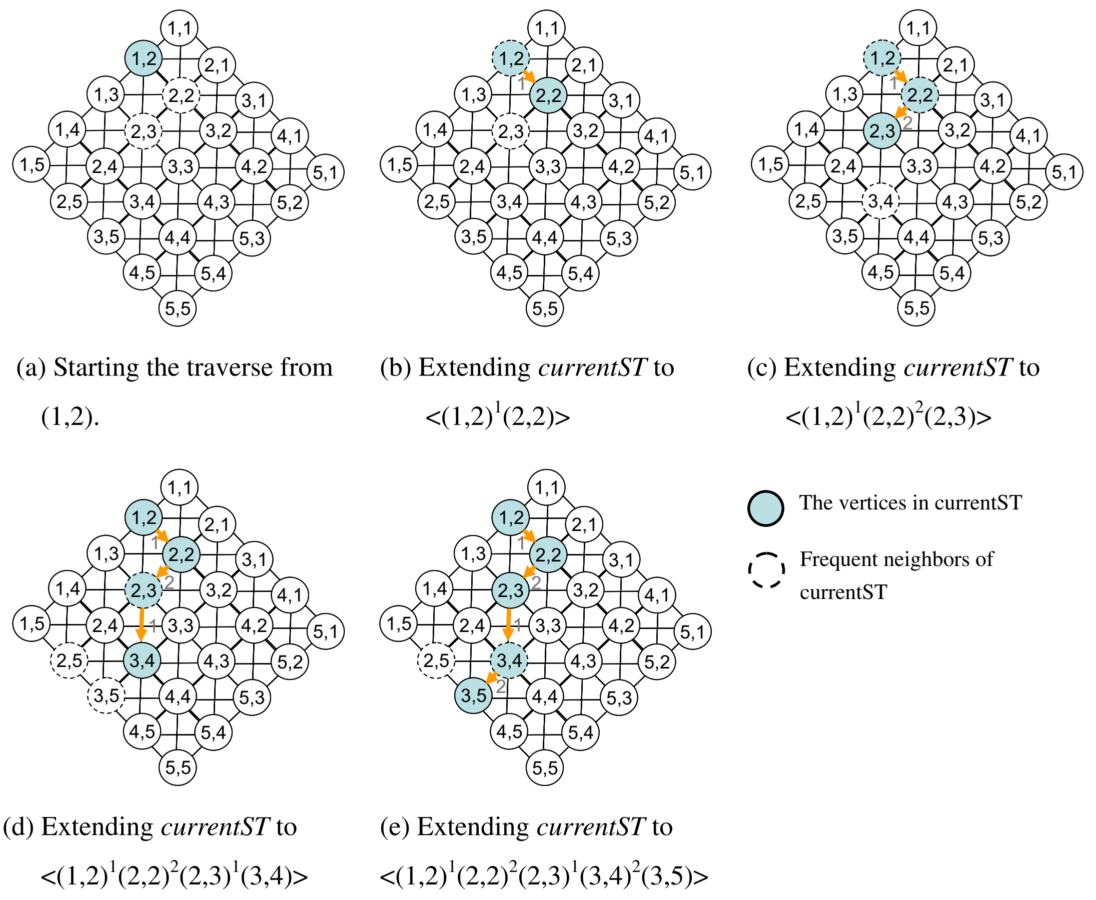
Fig. 4. The traverse procedure.

In the traverse procedure, we start traversing the mapping graph from the last vertex of *currentST*, which is denoted as *s* in step 1 of Fig. 4. For each level of the recursion, we append a vertex with a time span to *currentST*. We use the trajectories in *currentTI* and utilize the adjacency property to extend *currentST* by appending a frequent neighbor to *currentST*. In steps 2–17 of Fig. 4, we search the frequent neighbors of *currentST* clockwise to extend *currentST*. The vertex to be traversed is denoted as a_k in step 3. All trajectories passing through a_k are recorded in a_k's TI-list.

In steps 6–10, for each trajectory *t* in both *currentTI* and a_k's TI-list, we check if the time span between a_k and *s* in *t* is equal to *l*. If this is the case, *t* is inserted into *newTI*. In step 11, we count the number of trajectories in *newTI* and check if the support of the extended pattern is not less than the minimum support threshold, where the extended pattern is formed by appending *currentST* to a_k, and the time span between *s* and a_k is *l*. If this is the case, the extended pattern is frequent. In step 12, *currentST* is updated and added to *allST* in step 13. Then, we go to the next level and the trajectories in the *newTI* are used to further extend *currentST*. In step 14, the traverse procedure is called to further extend *currentST*. When the traverse procedure has finished, we have all the frequent patterns starting from a specific vertex.

Let us consider Example 1 again. After obtaining six TI-lists as shown in Fig. 2, we start the traverse procedure and the frequent pattern starting from vertex (1,2) is illustrated in Fig. 5, where the gray circles are the vertices already in *currentST*, the numbers beside edges denote the time spans, and the dotted circles are the frequent neighbors of *currentST*. In Fig. 5a, we first traverse the graph from vertex (1,2). The trajectories passing through vertex (1,2) are stored in its TI-list, and the support of the vertex is 2/3. Thus, we add $\langle(1,2)\rangle$ to *allST* and set *currentST* to $\langle(1,2)\rangle$. Then, we search the frequent neighbors of $\langle(1,2)\rangle$ and find that the support of the trajectories containing pattern $\langle(1,2)^1(2,2)\rangle$ is equal to 50%. Thus, *currentST* is extended to $\langle(1,2)^1(2,2)\rangle$, as shown in Fig. 5b. Next, we continue to extend *currentST* by using the adjacent vertex (2,2) and find that the support of $\langle(1,2)^1(2,2)^1(2,3)\rangle$ is equal to 0, and the support of $\langle(1,2)^1(2,2)^2(2,3)\rangle$ is equal to 50%. Thus, *currentST* is extended to $\langle(1,2)^1(2,2)^2(2,3)\rangle$, which is added to *allST* as shown in Fig. 5c. Similarly, we can extend *currentST* to $\langle(1,2)^1(2,2)^2(2,3)^1(3,4)\rangle$ as shown in Fig. 5d and then to $\langle(1,2)^1(2,2)^2(2,3)^1(3,4)^2(3,5)\rangle$ as shown in Fig. 5e. Both frequent patterns are added to *allST*. Since we cannot find any vertex to extend *currenST*, we backtrack to vertex (3,4) and check if we can find a vertex to extend *currentST*. As we cannot find any vertex to extend *currenST* in this trajectory, we backtrack to vertices (2,3), (2,2), (1,2) and find that no vertex can be used to extend *currentST*.

Starting from vertex (2,2), we can find the following frequent patterns: $\langle(2,2)\rangle$, $\langle(2,2)^3(2,3)\rangle$, $\langle(2,2)^3(2,3)^3(3,4)\rangle$, $\langle(2,2)^3(2,3)^3(3,4)^2(2,5)\rangle$, $\langle(2,2)^2(2,3)\rangle$, $\langle(2,2)^2(2,3)^1(3,4)\rangle$, and $\langle(2,2)^2(2,3)^1(3,4)^2(3,5)\rangle$. Similarly, starting from vertex (2,3), we can find $\langle(2,3)\rangle$, $\langle(2,3)^1(3,4)\rangle$, $\langle(2,3)^3(3,4)\rangle$, $\langle(2,3)^1(3,4)^2(3,5)\rangle$ and $\langle(2,3)^3(3,4)^2(2,5)\rangle$; and starting from vertex (3,4), we can find $\langle(3,4)\rangle$, $\langle(3,4)^2(3,5)\rangle$ and $\langle(3,4)^2(2,5)\rangle$. Finally, we find that $\langle(2,5)\rangle$ and $\langle(3,5)\rangle$ are also frequent patterns.



(a) Starting the traverse from (1,2).

(b) Extending *currentST* to $<(1,2)^1(2,2)>$

(c) Extending *currentST* to $<(1,2)^1(2,2)^2(2,3)>$

(d) Extending *currentST* to $<(1,2)^1(2,2)^2(2,3)^1(3,4)>$

(e) Extending *currentST* to $<(1,2)^1(2,2)^2(2,3)^1(3,4)^2(3,5)>$

Fig. 5. The frequent patterns starting from vertex (1,2).

Since no more frequent patterns can be found, the algorithm is terminated. Hence, the frequent patterns mined in the database are: $\langle(1,2)\rangle$, $\langle(2,2)\rangle$, $\langle(2,3)\rangle$, $\langle(3,4)\rangle$, $\langle(2,5)\rangle$, $\langle(3,5)\rangle$, $\langle(1,2)^1(2,2)\rangle$, $\langle(2,2)^2(2,3)\rangle$, $\langle(2,2)^3(2,3)\rangle$, $\langle(2,3)^1(3,4)\rangle$, $\langle(2,3)^3(3,4)\rangle$, $\langle(3,4)^1(3,5)\rangle$, $\langle(3,4)^2(2,5)\rangle$, $\langle(1,2)^1(2,2)^2(2,3)\rangle$, $\langle(2,2)^2(2,3)^1(3,4)\rangle$, $\langle(2,2)^3(2,3)^3(3,4)\rangle$, $\langle(2,3)^1(3,4)^2(3,5)\rangle$, $\langle(2,3)^3(3,4)^2(2,5)\rangle$, $\langle(1,2)^1(2,2)^2(2,3)^1(3,4)\rangle$, $\langle(2,2)^3(2,3)^3(3,4)^2(2,5)\rangle$, $\langle(2,2)^2(2,3)^1(3,4)^2(3,5)\rangle$, and $\langle(1,2)^1(2,2)^2(2,3)^1(3,4)^2(2,5)\rangle$.

Lemma 1. *The time complexity of the traverse procedure is bounded by* $O(|D| * \sum_{f=1}^{w}(PS_f + \sum_{i=1}^{Z_f} S_{f,i}))$, *where* $|D|$ *is the number of trajectories in the database, wis the number of frequent patterns starting from a specific vertex v,* PS_f *is the support of a frequent pattern* P_f *starting from v,* $1 \leqq f \leqq w$, Z_f *denotes the number of frequent neighbors of* P_f, *and* $S_{f,i}$ *is the support of a frequent neighbor of* P_f, $1 \leqq i \leqq Z_f$.

Theorem 2. *The time complexity of the GBM algorithm is bounded by* $O(|D| * |V| + |D| * \sum_{f=1}^{r}(PS_f + \sum_{i=1}^{Z_f} S_{f,i}))$, *where* $|D|$ *is the number of trajectories in the database,* $|V|$ *is the number of vertices in the mapping graph, r is the number of frequent patterns in the database,* PS_f *is the support of a frequent pattern* P_f, $1 \leqq f \leqq r$, Z_f *denotes the number of frequent neighbors of* P_f, *and* $S_{f,i}$ *is the support of a frequent neighbor of* P_f, $1 \leqq i \leqq Z_f$.

Let α be the average support of all frequent vertices, and β be the average support of all frequent patterns in the database. The time complexity of the GBM algorithm can also be denoted by $O(|D| * |V| + |D| * \sum_{f=1}^{r}(PS_f + \sum_{i=1}^{Z_f} S_{f,i})) = O(|D| * |V| + |D| * (r * \beta + r * Z_f * \alpha)) = O(|D| * |V| + |D| * r * \alpha) = O|D| * (|V| + r * \alpha)$ since $Z_f \leqq 8$, and $\beta \leqq \alpha$.

5. Performance evaluation

We conducted experiments to compare the proposed method with the modified MP (Moving Pattern mining) algorithm [9] (hereafter MP) and the modified PrefixSpan algorithm (hereafter PrefixSpan).

The original PrefixSpan algorithm is designed to mine sequential patterns over sequence databases and not trajectory patterns over trajectory databases, where time is not explicit as it is in a trajectory. Thus, the modified PrefixSpan algorithm needs to consider how to embed the time constraint into the mining process. That is, the time span between any two adjacent points in a pattern should be not greater than the maximum time span threshold. To embed the time constraint into the

Table 2
Parameters used to generate synthetic datasets.

Parameter	Description	Setting
D	The number of trajectories (K)	50–300
e	The length of the reference space	30–70
AL	The average length of the trajectories	10–100
MI	The maximum time span	30
F	The number of potential frequent patterns	1000
PL	The average length of the potential frequent patterns	25

mining process, the modified PrefixSpan algorithm finds the frequent 1-patterns from the projected database of a prefix such that for each frequent 1-patterns found, it is adjacent to the last point of the prefix, and the time span between the prefix and the frequent 1-pattern is not greater than the maximum time span threshold. That is, the algorithm finds the frequent 1-patterns only from the first point of each trajectory in the projected database. Then, for each frequent 1-pattern found, it is appended to the prefix to generate a new prefix. For each newly generated prefix, its projected database is built and the mining process is recursively repeated until no more frequent patterns can be found.

The original MP algorithm [9], an Apriori-based algorithm, was proposed for mining frequent trajectory patterns among different geological zones, where the time span between any two adjacent zones is not incorporated in the patterns. We revise the MP algorithm to mine the frequent patterns proposed in this paper. First, we scan the database to find all frequent 1-patterns. Next, we generate all candidate $(k + 1)$-patterns by combining any two joinable frequent k-patterns together, $k \geqslant 1$. Two frequent k-patterns are joinable if both k-patterns share the same $k - 1$ points and the time spans between any two adjacent points of these $k - 1$ points are the same. After generating all candidate $(k + 1)$-patterns, we scan the database once to count the support for each candidate and determine whether the candidate is frequent or not. Let $k = k + 1$. We repeat the steps described above until no more frequent patterns can be found.

5.1. Experimental environment

We conducted the experiments on both synthetic and real datasets to evaluate the performance of the GBM, PrefixSpan and MP algorithms. The synthetic data generator is similar to the one used by Agrawal et al. [1] with some modifications, since each transaction is a trajectory, not a sequence of itemsets. We generate the synthetic datasets as follows.

First, we assume the reference space is a square. Let the length of the reference space be e. Also, let MI be the maximum time span. Next, we generate F potential frequent patterns. To generate a potential frequent pattern, we randomly choose a point in the reference space as its starting point. To extend the pattern, we use a uniform distribution to choose a number between 1 and MI, which is the time span between the last point of the pattern and the newly generated point. Then, we randomly pick an adjacent point of the last point of the pattern to extend it. This process continues until the number of points in the pattern is equal to the predefined length which is determined by a Poisson distribution with mean PL. Finally, we use the potential frequent patterns to generate a synthetic dataset based on the parameters listed on Table 2.

The real dataset, which was compiled by the Institute of Information Science, Academia Sinica,[1] contains the trajectories of players in the PC online RPG game, "Angel Love Online". There are 749 trajectories of game players, where the time span between any two adjacent points is rounded to the nearest ten. The size of the game's reference space is 25 in length and 15 in width.

All the experiments were performed on a PC with an Intel Core 2 Duo 6300 CPU @ 1.86 GHz and 1 GB main memory, running on Microsoft Windows XP Professional. All the methods were implemented in Microsoft Visual C++ 6.0.

5.2. Experimental results and analyses

In this section, we compare the GBM algorithm with the MP and PrefixSpan algorithms by varying one parameter at a time, while maintaining the other parameters at the default values as shown in Table 2, where the default value of each parameter is fixed at the middle value except that the support is fixed at 1%.

The first experiment is conducted by using different minimum support thresholds under the condition $D150e50AL50\theta15$, where $D150$ means that the number of trajectories in the dataset is 150 K, $e50$ means that the length of the reference space is 50, $AL50$ means that the average length of the trajectories is 50, and $\theta15$ means that the maximum time span threshold is 15. The second experiment is conducted by using various numbers of trajectories under the condition $e50AL50S1\%\theta15$, where $S1\%$ means that the minimum support threshold is 1%. The third experiment is conducted by using trajectories with different average lengths under the condition $D150e50S1\%\theta15$. The fourth experiment is conducted by using different maximum time spans under the condition $D150e50AL50S1\%$. The fifth experiment is conducted by using reference spaces of different lengths under the condition $D150S1\%AL50\theta15$. The final experiment is conducted to evaluate the memory usage by using different minimum support thresholds under the condition $D150e50AL50\theta15$ and using trajectories with different average lengths under the conditions $D150e50S1\%\theta15$. The experimental results are shown in Figs. 6–11.

[1] http://mmnet.iis.sinica.edu.tw/~ktchen/angellove/.

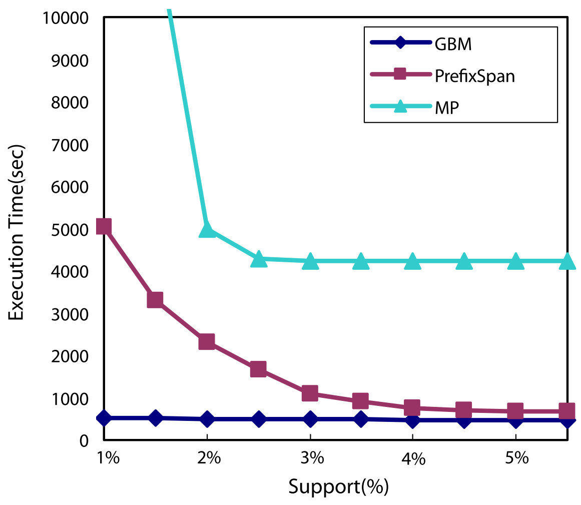


Fig. 6. Execution time vs. minimum support threshold.

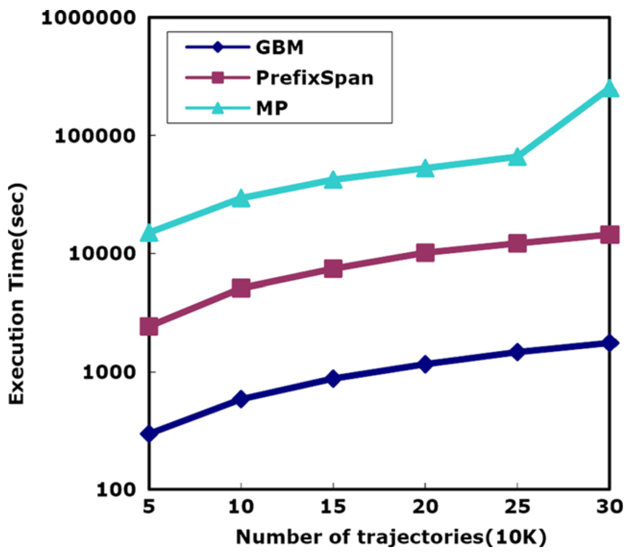


Fig. 7. Execution time vs. number of trajectories.

Fig. 6 shows the execution time versus the minimum support thresholds for the GBM, PrefixSpan, and MP algorithms. As the minimum support threshold decreases, the execution time of the three algorithms increases because the number of frequent patterns increases substantially. To mine frequent patterns, the PrefixSpan algorithm needs to project the database many times and the MP algorithm generates a large number of candidates; however, the GBM algorithm can localize support counting and pattern extension to a small number of TI-lists. As a result, the GBM algorithm runs about 2–7 times faster than the PrefixSpan algorithm and 8–50 times faster than the MP algorithm.

Fig. 7 shows the execution time versus the number of trajectories in the database, where the size of the datasbase varies from 50 K to 300 K. The GBM algorithm runs about 7–8 times faster than the PrefixSpan algorithm and 60–170 times faster than the MP algorithm. Although the execution time of the three algorithm increases linearly as the number of trajectories increases, the execution time of the MP and Prefixspan algorithm grows more sharply than that of the GBM algorithm.

Fig. 8 illustrates the execution time versus the average length of trajectories for both algorithms, where the average length of trajectories varies from 10 to 100. As the average length of the trajectories increases, more and longer frequent patterns will be generated. Thus, the execution time of the three algorithms increases. However, the GBM algorithm runs about 4–7 times faster than the PrefixSpan algorithm and about 15–500 times faster than the MP algorithm.

Fig. 9 presents the execution time versus the maximum time span. As the maximum time span increases, the execution time of the three algorithms increases and the number of frequent patterns increases substantially. The GBM algorithm runs about 5–7 times faster than the PrefixSpan algorithm and about 40–600 times faster than the MP algorithm. The larger the

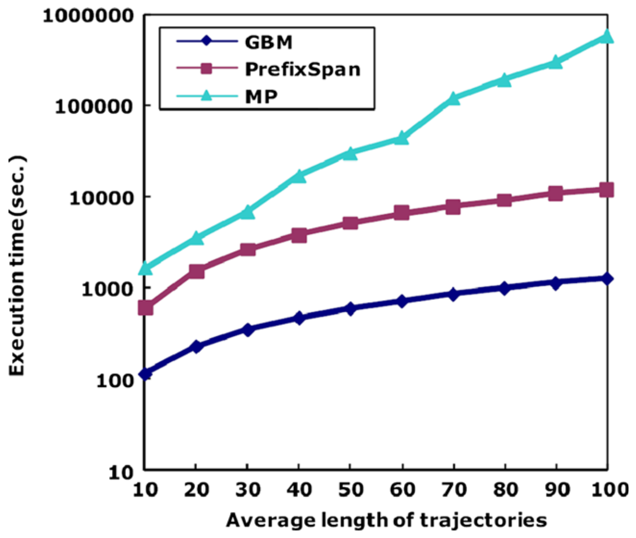


Fig. 8. Execution time vs. average length of trajectories.

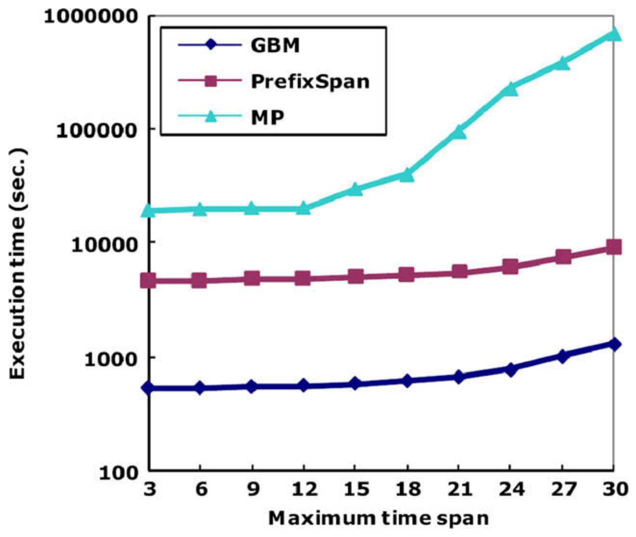


Fig. 9. Execution time vs. maximum time span.

maximum time span, the more projected databases are generated; hence, the PrefixSpan algorithm needs to scan the projected databases many times. For the MP algorithm, the larger the maximum time span, the more candidates are generated. Thus, its execution time increases quickly as the maximum time span increases. In contrast, the GBM algorithm can localize support counting and pattern extension to a small number of TI-lists. Moreover, the GBM algorithm utilizes the adjacency property to reduce the search space. Thus, it outperforms the PrefixSpan and MP algorithms.

Fig. 10 illustrates the execution time versus the length of the reference space. As the length of the reference space increases, the execution time of the three algorithms increases. The GBM algorithm runs about 6–10 times faster than the PrefixSpan algorithm and about 45–80 times faster than the MP algorithm. The larger the length of the reference space is, the sparser the trajectories are. The sparser the trajectories, the fewer frequent trajectory patterns. Therefore, the execution time decreases while the length of the reference space increases.

Fig. 11 shows the memory used by the TI-lists and currentTI-list during the mining process under different minimum support thresholds and different average trajectory lengths. Since the GBM algorithm recursively calls the traverse procedure, we sum up the memory used by currentTI for each recursion and the memory used by TI-lists. We can observe that the memory used increases when the average length of trajectories increases or the minimum support threshold decreases.

We also used a real dataset to evaluate the performance of the algorithms. The performance of each algorithm on the real dataset is quite similar to that on the synthetic datasets.

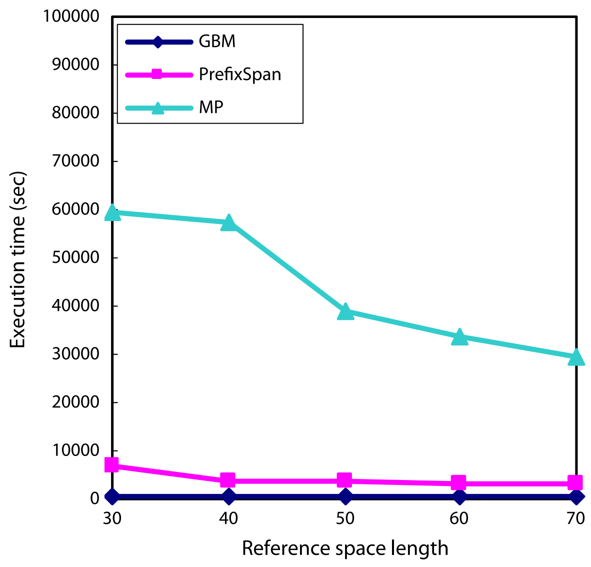


Fig. 10. Execution time vs. length of reference space.

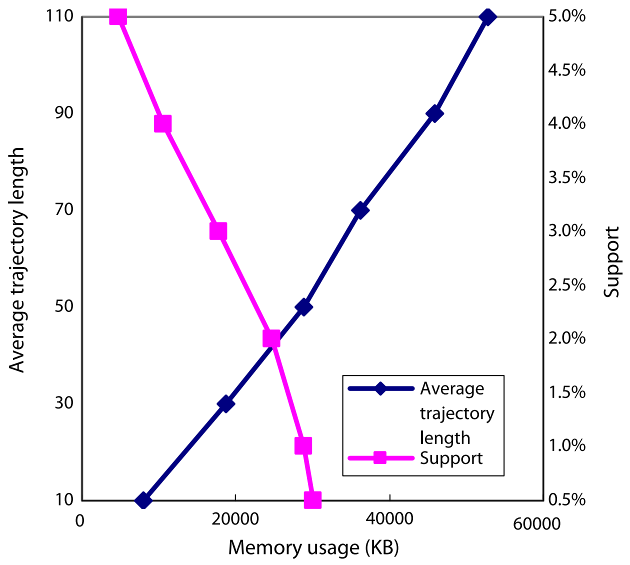


Fig. 11. Memory usage.

In addition, Fig. 12 illustrates three frequent patterns, which are denoted by the solid, dotted, and dashed lines on the map, respectively. The first frequent pattern denoted by a solid line is $\langle(3,12)^{210}(2,12)^{400}(3,12)^{200}(2,12)^{420}(3,12)^{210}(2,12)^{400}(3,12)^{190}(2,12)^{390}(3,12)^{210}(2,12)^{410}(3,12)\rangle$. That means it takes the players around two hundred seconds to defeat monsters at (3,12), and then around four hundred seconds at (2,12). By defeating monsters, the players can obtain money and some valuable items. After the monsters are defeated, they will be reborn. Thus, the players keep moving between (3,12) and (2,12) to defeat monsters and obtain more money and valuable items.

The second pattern denoted by a dotted line is $\langle(5,14)^{300}(6,13)^{0}(7,13)^{0}(8,12)^{0}(9,12)^{0}(9,11)^{0}(10,10)^{0}(11,10)\rangle$. Since the time span is rounded to the nearest 10, "0" means that the time span between two adjacent points is less than 5 s. This pattern denotes that the players spend around 300 s at (5,14) to hide from the monsters and recover their health points (vitality). The third pattern denoted by a dashed line is $\langle(11,9)^{0}(12,9)^{0}(13,9)^{0}(13,8)^{0}(14,8)^{0}(15,8)^{0}(15,7)^{0}(16,7)^{0}(17,7)^{0}(17,6)^{0}(18,6)^{0}(19,6)^{0}(20,6)^{0}(21,6)^{0}(22,7)^{30}(22,6)\rangle$. That means the players spend round 30 s to defeat the special monster (Acolyte of tribe) at (22,7) since the players must defeat the monster to accomplish their missions and obtain a special item to increase their power.

In summary, the experimental results show that the GBM algorithm outperforms the Apriori-based and PrefixSpan-based algorithms by more than one order of magnitude. The main reason is that the GBM algorithm can localize support counting

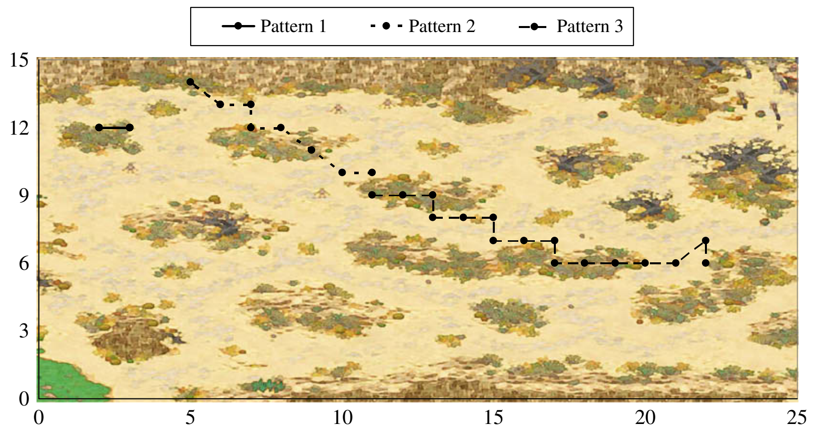


Fig. 12. Map of Angel Love Online.

and pattern extension to a small number of TI-lists. Moreover, the GBM algorithm utilizes the adjacency property to reduce the search space. Therefore, the GBM algorithm is more efficient and scalable than the PrefixSpan algorithm.

6. Conclusions and future work

In this paper, we have proposed an efficient graph-based mining (GBM) algorithm for mining frequent trajectory patterns. The algorithm comprises two phases. In the first phase, we transform all trajectories in the database into a mapping graph. Then, we create a TI-list for each vertex to record all trajectories that pass through the vertex. In the second phase, using the information recorded in TI-lists, the proposed algorithm traverses the mapping graph in a depth-first search manner to mine all frequent trajectory patterns.

By exploiting the TI-lists and the adjacency property to mine the frequent trajectory patterns, the search space can be reduced substantially. Therefore, our proposed algorithm is more efficient and scalable than the Apriori-based and PrefixSpan-based algorithms. The experimental results show that the GBM algorithm outperforms the Apriori-based and PrefixSpan-based algorithms by more than one order of magnitude.

Although we have shown that the GBM algorithm can mine frequent trajectory patterns efficiently, there are still some issues to be addressed in future research. First, by using the adjacency property to mine frequent patterns in the proposed algorithm, we cannot mine the patterns formed by non-adjacent points. For example, if a group of people move from the beginning point *A* to the ending point *B* by different paths, our proposed algorithm cannot mine the pattern containing the beginning and ending points since *A* is not adjacent to *B*. It is worth extending the proposed algorithm to mine such patterns. Second, it is worth extending the proposed algorithm to deal with the case when currentTI lists are too large to fit in main memory. One possible way to deal with such a problem is to divide the reference space into several partitions such that the currentTI lists in each partition can be fitted in main memory. Thus, we can mine the frequent patterns for each partition and then merge these frequent patterns mined together. Third, the patterns mined by the proposed algorithm are represented by a sequence of adjacent points. It is worth simplifying the pattern representation where a pattern can be denoted by a sequence of segments such as the trajectory representation used in [3]. Finally, the proposed algorithm could further be used for other applications such as mobile advertisements, shoppers' trajectory analysis, and animal trajectory analysis to find meaningful patterns.

Acknowledgements

The authors are grateful to the anonymous referees for their helpful comments and suggestions. This research was supported in part by the National Science Council of Republic of China under Grant No. NSC 97-2410-H-002-117.

Appendix A. Here, we present the proofs of Theorems 1 and 2.

Theorem 1. *The time complexity of the construction algorithm is bounded by* $O(e^2 + \sum_{i=1}^{u} PL_i)$*, where* PL_i *is the length of trajectory* T_i*, the length and width of the reference space are equal toe, and u is the number of trajectories in the database.*

Proof. Let the trajectories in the database be $T_1, T_2, \ldots, T_u$, and the length of the trajectory T_i be PL_i, $1 \leqq i \leqq u$. The time complexity of steps 1–6 is obviously bounded by $O(e^2)$. In steps 7–11, the information about a trajectory is added to the corresponding TI-lists. Since the time complexity of adding the information about the trajectory T_i to the TI-lists in steps 8–10 is bounded by $O(PL_i)$, the time complexity of steps 7–11 is bounded by $O(\sum_{i=1}^{u} PL_i)$. Also, the time complexity of steps 12–16 is bounded by $O(e^2)$. Therefore, the time complexity of the construction algorithm is bounded by $O(e^2 + \sum_{i=1}^{u} PL_i)$.

Before analyzing the time complexity of the GBM algorithm, we consider the time complexity of the traverse procedure. □

Lemma 1. *The time complexity of the traverse procedure is bounded by* $O(|D|^{*}\sum_{f=1}^{w}(PS_f + \sum_{i=1}^{Z_f} S_{f,i}))$, *where* $|D|$ *is the number of trajectories in the database, w is the number of frequent patterns starting from a specific vertex v,* PS_f *is the support of a frequent pattern* P_f *starting from v,* $1 \leqq f \leqq w$, Z_f *denotes the number of frequent neighbors of* P_f, *and* $S_{f,i}$ *is the support of a frequent neighbor of* P_f, $1 \leqq i \leqq Z_f$.

Proof. The traverse procedure recursively finds all frequent patterns starting from a specific vertex v. Let w be the number of frequent patterns starting from a specific vertex v and the frequent patterns starting from v be $P_1, P_2, \ldots, P_w$, and the support of P_f be PS_f, $1 \leqq f \leqq w$. Let Z_f be the number of frequent neighbors of P_f, and the frequent neighbors of P_f be $N_{f,1}, N_{f,2}, \ldots, N_{f,Zf}$. Let $S_{f,i}$ be the support of $N_{f,i}$, $1 \leqq i \leqq Z_f$.

Let us start the traverse procedure from a specific vertex v and *currentST* be P_f. The time complexity of finding the trajectory t appearing in both *currentTI* and a_k's TI-list with time span = l in steps 6–10 is bound by $O(m + n)$ since both *currentTI*and a_k's TI-list are sorted, where m and n are the length of both lists, respectively. If the number of trajectories in *newTI* is not less than $\sigma^{*}|D|$ in step 11, it means *currentST*is extended by appending P_f witha_k, where a_k is a frequent neighbor of P_f. Thus, the time complexity of extending *currentST* is bounded by $O(|D|^{*}(PS_f + S_{f,i}))$, where $|D|^{*}PS_f$ is the number of trajectories in *currentST* and $|D|^{*}S_{f,i}$ is the number of trajectories in a_k's TI-list. Otherwise, *currentST* extended by appending P_f with a_k is infrequent and the time complexity of this case is bounded by $O(|D|^{*}(PS_f+S_{f,j}))$, too. That is, the time complexity of steps 6–10 is bounded by $O(| D | *(PS_f + \sum_{i=1}^{Z_f} S_{f,i})$, where $S_{f,i}$ is the support of a frequent neighbor of P_f and Z_f is the number of frequent neighbors of P_f.

There are two for-loops containing steps 6–15, where the outer loop of steps 2–17 is executed *neighbors* times and the inner loop of steps 5–16 is executed θ times. The time complexity of steps 12–13 is bounded by $O(1)$. In step 14, the traverse procedure is recursively called to extend *currentST* until all frequent patterns starting from v are found. That is, the traverse procedure is called w times, where w is the number of frequent patterns starting from v. Thus, the time complexity of the traverse procedure is bounded by $O(|D| * neighbors * \theta * \sum_{f=1}^{w}(PS_f + \sum_{i=1}^{Z_f} S_{f,i}))$. However, a vertex has at most eight adjacent vertices. That is, $neighbors \leqq 8$. Moreover, θ is a constant. Therefore, the time complexity of the traverse procedure is bounded by $O(|D| * \sum_{f=1}^{w}(PS_f + \sum_{i=1}^{Z_f} S_{f,i}))$. □

Theorem 2. *The time complexity of the GBM algorithm is bounded by* $O(|D| * |V| + |D| * \sum_{f=1}^{r}(PS_f + \sum_{i=1}^{Z_f} S_{f,i}))$, *where* $|D|$ *is the number of trajectories in the database,* $|V|$ *is the number of vertices in the mapping graph, r is the number of frequent patterns in the database,* PS_f *is the support of a frequent pattern* P_f, $1 \leqq f \leqq r$, Z_f *denotes the number of frequent neighbors of* P_f, *and* $S_{f,i}$ *is the support of a frequent neighbor of* P_f, $1 \leqq i \leqq Z_f$.

Proof. The traverse procedure is called for each vertex in the mapping graph to mine all frequent patterns starting from the vertex in the GBM algorithm. For a vertex v, the time complexity of step 3 is bounded by $O(1)$. The time complexity of step 4 is bounded by O(the number of trajectories containing *currentST*), which is bounded by $O(|D|)$. According to Lemma 1, the time complexity of step 5 is bounded by $O(|D| * \sum_{f=1}^{w}(PS_f + \sum_{i=1}^{Z_f} S_{f,i}))$, where w is the number of frequent patterns starting from v, PS_f is the support of a frequent pattern P_f starting from v, $1 \leqq f \leqq w$, Z_f denotes the number of frequent neighbors of P_f, and $S_{f,i}$ is the support of a frequent neighbor of P_f, $1 \leqq i \leqq Z_f$. Since steps 2–6 are executed once for each vertex in the mapping graph, the time complexity of the GBM algorithm is equal to the summation of the time complexity for each vertex. Therefore, the time complexity of the GBM algorithm is bounded by $O(|D| * |V| + |D| * \sum_{f=1}^{r}(PS_f + \sum_{i=1}^{Z_f} S_{f,i}))$, where r is the total number of frequent patterns in the database. □

References

[1] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1993, pp. 207–216.

[2] J, Ayres, J.E. Gehrke, T. Yiu, J. Flannick, Sequential pattern mining using a bitmap representation, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, pp. 429–435.

[3] A. Brilingaite, C.S. Jensen, N. Zokaite, Enabling routes as context in mobile services, in: Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems, 2004, pp. 127–136.

[4] H. Cao, N. Mamoulis, D.W. Cheung, Mining frequent spatio-temporal sequential patterns, in: Proceedings of the 5th IEEE International Conference on Data Mining, 2005, pp. 82–89.

[5] H. Cao, N. Mamoulis, D.W. Cheung, Discovery of collocation episodes in spatiotemporal data, in: Proceedings of the Sixth IEEE International Conference on Data Mining, 2006, pp. 823–827.

[6] H. Cao, N. Mamoulis, D.W. Cheung, Discovery of periodic patterns in spatiotemporal sequence, IEEE Transactions on Knowledge and Data Engineering 19 (4) (2007) 453–467.

[7] Y.L. Chen, Y.H. Hu, Constraint-based sequential pattern mining: the consideration of recency and compactness, Decision Support Systems 42 (2) (2006) 1203–1215.

[8] D.Y. Choi, Personalized local internet in the location-based mobile web search, Decision Support Systems 43 (1) (2007) 31–45.

[9] J.D. Chung, O.H. Paek, J.W. Lee, K.H. Ryu, Temporal moving pattern mining for location-based service, in: Proceedings of the 13th International Conference on Database and Expert Systems Applications, 2002, pp. 481–490.

[10] M. Garofalakis, R. Rastogi, K. Shim, Mining sequential patterns with regular expression constraints, IEEE Transactions on Knowledge and Data Engineering 14 (3) (2002) 530–552.

[11] F. Giannotti, M. Nanni, D. Pedreschi, F. Pinelli, Mining sequences with temporal annotations, in: Proceedings of the ACM Symposium on Applied Computing, 2006, pp. 593–597.
[12] K. Gade, J. Wang, G. Karypis, Efficient closed pattern mining in the presence of tough block constraints, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004, pp. 138–147.
[13] G. Grahne, L.V.S. Lakshmanan, X. Wang, Efficient mining of constrained correlated sets, in: Proceedings of IEEE International Conference on Data Engineering, 2000, pp. 512–521.
[14] E. Gudes, E. Shimony, N. Vanetik, Discovering frequent graph patterns using disjoint paths, IEEE Transactions on Knowledge and Data Engineering 18 (11) (2006) 1441–1456.
[15] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.C. Hsu, Mining frequent patterns without candidate generation, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2000, pp. 1–12.
[16] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.C. Hsu, FreeSpan: frequent pattern-projected sequential pattern mining, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, pp. 355–359.
[17] T. Hu, S.Y. Sung, H. Xiong, Q. Fu, Discovery of maximum length frequent itemsets, Information Sciences 178 (1) (2008) 69–87.
[18] J. Huan, W. Wang, J. Prins, Efficient mining of frequent subgraphs in the presence of isomorphism, in: Proceedings of the IEEE International Conference on Data mining, 2003, pp. 549–552.
[19] Y. Huang, H. Xiong, W. Wu, P. Deng, Z. Zhang, Mining maximal hyperclique pattern: a hybrid search strategy, Information Sciences 177 (3) (2007) 703–721.
[20] S.Y. Hwang, Y.H. Liu, J.-K. Chiu, E.-P. Lim, Mining mobile group patterns: a trajectory-based approach, in: Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2005, pp. 713–718.
[21] A. Inokuchi, T. Washio, H. Motoda, An Apriori-based algorithm for mining frequent substructures from graph data, in: Proceedings of the European Conference on Principles and Practice of Knowledge in Databases, 2000, pp. 13–23.
[22] R. Jin, C. Wang, D. Polshakov, S. Parthasarathy, G. Agarwal, Discovery frequent topological structures from graph datasets, in: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2005, pp. 606–611.
[23] M. Kuramochi, G. Karypis, Frequent subgraph discovery, in: Proceedings of the IEEE International Conference on Data Mining, 2001, pp. 313–320.
[24] A.J.T. Lee, R.W. Hong, W.M. Ko, W.K. Tsao, H.H. Lin, Mining spatial association rules in image databases, Information Sciences 177 (7) (2007) 1593–1608.
[25] A.J.T. Lee, W.C. Lin, C.S. Wang, Mining association rules with multi-dimensional constraints, The Journal of Systems and Software 79 (1) (2006) 79–92.
[26] A.J.T. Lee, Y.T. Wang, Efficient data mining for calling path patterns in GSM networks, Information Systems 28 (8) (2003) 929–948.
[27] M. Leleu, C. Rigotti, J.F. Boulicaut, G. Euvrard, GO-SPADE: mining sequential patterns over datasets with consecutive repetitions, in: Proceedings of International Conference on Machine Learning and Data Mining, 2001, pp. 293–306.
[28] R. Ng, L.V.S. Lakshmanan, J. Han, A. Pang, Exploratory mining and pruning optimizations of constrained association rules, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1998, pp. 13–24.
[29] S. Orlando, R. Perego, C. Silvestri, A new algorithm for gap constrained sequence mining, in: Proceedings of the ACM Symposium on Applied computing, 2004, pp. 540–547.
[30] J. Pei, J. Han, Can we push more constraints into frequent pattern mining? in: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, pp. 350–354.
[31] J. Pei, J. Han, L.V.S. Lakshmanan, Mining frequent itemsets with convertible constraints, in: Proceedings of IEEE International Conference on Data Engineering, 2001, pp. 433–442.
[32] J. Pei, J. Han, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.C. Hsu, PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth, in: Proceedings of the IEEE International Conference on Data Engineering, 2001, pp. 215–224.
[33] R. Srikant, R. Agrawal, Mining sequential patterns: generalizations and performance improvements, in: Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology, 1996, pp. 3–17.
[34] R. Srikant, Q. Vu, R. Agrawal, Mining association rules with item constraints, in: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1997, pp. 67–73.
[35] I. Tsoukatos, D. Gunopulos, Efficient mining of spatiotemporal patterns, in: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases, 2001, pp. 425–442.
[36] C.Y. Wang, S.S. Tseng, T.P. Hong, Flexible online association rule mining based on multidimensional pattern relations, Information Sciences 176 (12) (2006) 1752–1780.
[37] X. Yan, J. Han, gSpan: graph-based substructure pattern mining, in: Proceedings of International Conference on Data Mining, 2002, pp. 721–724.
[38] J.X. Yu, Z. Chong, H. Lu, Z. Zhang, A. Zhou, A false negative approach to mining frequent itemsets from high speed transactional data streams, Information Sciences 176 (14) (2006) 1986–2015.
[39] U. Yun, Efficient mining of weighted interesting patterns with a strong weight and/or support affinity, Information Sciences 177 (17) (2007) 3477–3499.
[40] U. Yun, A new framework for detecting weighted sequential patterns in large sequence databases, Knowledge-Based Systems 21 (2) (2008) 110–122.
[41] M.J. Zaki, SPADE: an efficient algorithm for mining frequent sequences, Machine Learning 11 (5) (2001) 31–60.