International Journal of Computational Intelligence Systems, Vol.3, No. 3 (September, 2010), 343-356

Parallel Sequential Pattern Mining of Massive Trajectory Data*

Shaojie Qiao [1]†, Tianrui Li [1], Jing Peng [2], Jiangtao Qiu [3]

[1] *School of Information Science and Technology, Southwest Jiaotong University*
No. 111, Erhuanlu Beiyiduan, Chengdu, Sichuan 610031, China
E-mail: qiaoshaojie@gmail.com

[2] *Department of Science and Technology, Chengdu Public Security Bureau*
No. 136, Wenwu Road, Chengdu, Sichuan 610017, China
E-mail: pj@tfol.com

[3] *School of Information, Southwestern University of Finance and Economics*
No. 55, Guanghuacun Road, Chengdu, Sichuan 610074, China
E-mail: Jiangtaoqiu@gmail.com

Received: 23-12-2009
Accepted: 31-05-2010

Abstract

The trajectory pattern mining problem has recently attracted much attention due to the rapid development of location-acquisition technologies, and parallel computing essentially provides an alternative method for handling this problem. This study precisely addresses the problem of parallel mining of trajectory sequential patterns based on the newly proposed concepts with regard to trajectory pattern mining. We propose an efficient and effective **p**ara**l**lel seq**u**ential pa**tte**rns mining (plute) algorithm that includes three essential techniques: prefix projection, data parallel formulation, and task parallel formulation. Firstly, the prefix projection technique is used to decompose the search space as well as greatly reduce the candidate trajectory sequences. Secondly, the data parallel formulation decomposes the computations associated with counting the support of trajectory patterns. Thirdly, the task parallel formulation employs the MapReduce programming model to assign the computations across a set of machines in a scalable and easy-to-use fashion. Based on the properties of parallel trajectory sequences, item pruning and sequence pruning strategies are applied to further prune the candidate sequences. Extensive experiments are conducted to evaluate the performance of **plute** in terms of parallel computing time and communication cost among processors. Experimental results show that plute outperforms the previously proposed parallel mining strategy (PartSpan) in mining massive trajectory data.

Keywords: parallel computing; trajectory sequential patterns; prefix projection; data parallel formulation; task parallel formulation; massive trajectory data

*This work is supported by the National Natural Science Foundation of China under Grant Nos. 60773169, 60873108; the National Science Foundation for Post-doctoral Scientists of China under Grant No. 20090461346; the Fundamental Research Funds for the Central Universities under Grant No. SWJTU09CX035; the Education Ministry Youth Fund of Humanities and Social Science of China under Grant No. 09YJCZH101; the Youth Science and Technology Science of Sichuan under Grant No. 08JJ0109; the Youth Software Innovation Project of Sichuan Province under Grant Nos. 2007AA0032, 2007AA0028.
†Corresponding author.

1. Introduction

The growing advances in wireless communications and location aware techniques enable us to collect a large amount of trajectory data of moving objects. For example, Global Positions Systems (GPS) have been widely used to trace the instantaneous locations of traffic facilities and mobile devices. Telematics is another rapidly growing real-time techniques which can provide emergency roadside assistance, stolen vehicle tracking and automatic crash notification etc[1]. In addition, we can obtain ever-increasing amount of trajectory data which are time-stamped sequences of events by location-based facilities.

Discovery of frequent trajectory sequential patterns is an essential data mining task with broad applications as well as a challenging problem especially for massive data. The objective of sequential pattern mining is to discover frequent subsequences in a data set[2]. The time annotation in trajectory patterns is an important notion and works as a user-specified constraint to preprocess the input data into ordered sequences of events, or as a pruning mechanism to shrink the search space[3]. Importantly, the time annotated sequence can be used in a variety of areas, i.e., the discovery of motifs in DNS sequences, traffic tracking, the analysis of web log[4], crime hotspot detection[5], and animal movement extraction[1]. So, it is of great practical value to utilize the spatio-temporal information of trajectories to improve the computational efficiency which provide an opportunity to automatically discover the useful knowledge from the trajectory databases[6].

Parallel computing is a form of computation in which a large amount of calculations are carried out simultaneously, operating on the principle that large problems can be divided into smaller ones, which are then solved in paralle‡ There are several different kinds of parallel computing, e.g., bit-level, instruction level, data, and task parallelism. Parallelism has been employed mainly in high-performance computing, but interest in it has grown lately in computer architecture, especially in the form of multicore processors[7]. Parallel computing is an alternative solution for parallel sequence mining[8]. Serial algorithms cannot provide scalability when giving strict constraints on the data size and the performance especially for large databases[8]. The rapid development of multiprocessor systems provide more opportunities for use to develop efficient and effective parallel computing algorithms.

There are two commonly-used methods for utilizing multiple processors, i.e., distributed memory in which each processor has a private memory and shared memory in which each processor has a common memory[8]. Subsequently, a distributed-shared memory (DSM) architecture arises, which combine the best of the previous two techniques. A good parallel algorithm should be efficient in the data parallel and task parallel formulations, rather than more relying on the hardware or software share, which motivate us to develop a new trajectory sequential pattern mining algorithm based on parallel formulations.

In this paper, we make the following contributions.

1. We exactly address the problem of parallel trajectory sequential pattern mining and propose a new task parallel formulation approach based on MapReduce[9].

2. We propose a hybrid approach by combining the data parallel and the task parallel formulation techniques to decompose computations and assign the computations of mining trajectory sequential patterns to multiple processors with lower communication and computation cost.

3. We integrate the time constraint into the sequence items and use sequence pruning strategies to eliminate the candidate patterns based on the properties of trajectory patterns in parallel computing environment.

4. We perform extensive experiments to estimate the performance of the proposed parallel algorithm of discovering trajectory sequential patterns based on the parallel formulation techniques in terms of the parallel time and communication cost across multiple processors.

‡https://computing.llnl.gov/tutorials/parallel_comp/

The rest of the paper is organized as follows. Section 2 summarizes the related work on the trajectory sequential pattern mining. Section 3 formally addresses the problem statement associated with the trajectory sequential pattern mining and the essential concepts used in this paper. Section 4 analyzes the properties of parallel trajectory patterns. Section 5 details the parallel sequence mining of trajectory patterns. Included is the MapReduce based task parallel formulation approach and the pruning strategies. Section 6 presents and analyzes the experimental results. Finally, section 7 gives the concluding remarks and outlooks future research directions.

2. Related Work

In this section, we will introduce the related works that can be categorized into three research directions including sequential pattern mining, spatio-temporal sequence mining and parallel computing.

2.1. Sequential Pattern Mining

The problem of mining frequent sequential patterns can be defined over a sequential database D, and the item of each sequence is annotated with a timestamp, which determines the order of the elements in the sequence[10]. For instance, an item set s is denoted as $(s_1 s_2 \dots s_k)$ that contains k items. A sequence $a = a_1 \mapsto a_2 \mapsto \dots \mapsto a_m$, where a_i is an item set. If there exists a sequence $b = a_1 \mapsto a_2 \mapsto \dots \mapsto a_n$ satisfying that $\forall_{1 \leqslant k \leqslant n}, a_k \subseteq b_{i_k}$, where $1 \leqslant s_1 < \dots < s_m \leqslant n$, then we call a is a subsequence of b, denoted as $a \preceq b$. The support count of a, denoted as $Support(a)$, is the total number of sequences that contain a. Given a minimum support threshold min_sup, a is frequent if it occurs more than min_sup times, i.e., $Support(a) \geqslant min\_sup$[3].

Mining sequential patterns has been studied for several years. To our knowledge, Dietterich et al. were the first to systematically address the problem of discovering patterns in sequences of events. However, this work focuses on discovering the rules characterizing a sequence and is able to predict a plausible sequence continuation[11]. Then, Agrawal et al. proposed an algorithm for finding all sequential patterns, called AprioriAll[2]. However, the phase of transformation by replacing itemsets in each transaction is costly. In order to better handle the problem of transforming in massive data, Srikant proposed an influential approach for mining sequential patterns, called GSP[12], which is a generalized sequential pattern algorithm for mining all the frequent sequences without transforming the database. Another typical approach for mining sequential patterns is SPADE[13] that employs lattice search techniques and simple join operations. Essentially, SPADE needs only three scans over the database.

Several efficient sequential pattern mining algorithms has been proposed, among which the prefix-based approaches, i.e., FreeSpan[14] and PrefixSpan[15] are more efficient than the above methods. They use the prefix projection technique to reduce the size of projected databases. PrefixSpan is widely applied to discover the temporal-spatial patterns.

2.2. Spatio-temporal Sequence Mining

Spatio-temporal pattern mining has recently grown to be an active research topic, which helps understand the mobility-related phenomena[16]. In order to discover spatio-temporal patterns, Cao et al.[17] transform the spatio-temporal sequence into spatial regions around frequent line-segments and detect frequent regions in a heuristic way. An influential work was done by Giannot et al[3]. They proposed the trajectory pattern mining problem and employed an aggregated trajectory extraction method within an observed population of trajectories to mine spatial-temporal patterns[3]. Trajectory patterns are a spatio-temporal variant of the temporally-annotated sequences (TAS)[10], where the time dimension is considered. By extending the work on TAS-mining (especially suitable for weblog analysis), Giannotti et al.[3] proposed a density-based algorithm to find regions of interest. The mechanism of discovering *popular* region is to compute candidate places by selecting all minimal square regions that are visited relatively frequent. Moreover, Lee et al.[18] proposed a density-based line-segment clustering algorithm to discover the sub-trajectories simultaneously. However, all the previous approaches for discovering sequential patterns are serial algorithms. In case of a large sequence database with massive data, these

algorithms need multiple passes over the databases, which needs much calculation time. An alternative solution is the parallel technique that relies on multiprocessor system to handle this problem, because a parallel algorithm can be executed for a piece at a time on several processers that can utilize the collection of computer resources.

2.3. Parallel Algorithms

The parallel mining of sequential patterns has recently received increasing attention[8,19,20]. Shintanik et al.[19] proposed three parallel algorithm for mining sequential patterns. The hash-based partition sequential pattern mining algorithm (HPSPM) uses an intelligent method to partition the candidate sequences by using hash function and it was evaluated to be better than the other two approaches. An efficient parallel algorithm for discovering the sequential patterns in massive data is pSPADE[8]. Its nature is a parallel SPADE[13]. pSPADE decomposes the search space into multiple suffix-based classes and uses the dynamic load balance strategy to process the tasks independently without synchronized operations. An alternative method is the parallel tree projection algorithm[21]. The key idea behind this approach is to integrate the data parallel and task parallel formulations based on tree projection in order to distribute the computations into multiple processors in an accurate manner, i.e., assign the tasks in terms of the relative amount of workload associated with each sequential pattern. The recently typical work includes FDMSP[22] and DMGSP[23], which adopt the similar strategy to compress local frequent sequential patterns into a lexicographic sequence tree without translations of repeated prefixes.

The trajectory pattern mining problem is specific, since the time-stamps of the sequence is constrained with a specified time interval. For instance, there are several routes from p_1 to p_2. One time interval t_1 corresponding to a path is 10 minutes, and the other one t_2 is 15 minutes. Suppose a time tolerance τ equals 8 minutes, because $|t_1 - t_2| < \tau$, we treat these two routes as similar. Whereas, we have to take into consideration the minimum support threshold for eliminating the infrequent sequential patterns. In order to discover trajectory patterns from massive data, we have proposed a parallel sequential pattern mining algorithm, namely PartSpan, which combine effective data parallel and task parallel formulations to distribute the computations across multiple processors. However, the scalability and fault tolerance of the task parallel formulation in PartSpan have not been carefully considered. When a processor fails, the performance of parallel computing will drastically fall down. This limitation pushes us to propose a more robust and scalable task parallel formulation approach.

3. Problem Statement

The parallel mining of trajectory patterns is defined as discovering frequent trajectory patterns (i.e., FT-pattern) across multiple processors[3]. It is a preferable solution to the sequence mining problem[8]. In this section, we first introduce the preliminaries, and then formalize the trajectory pattern mining problem in a parallel computing environment.

Definition 1. (Trajectory pattern)[3] A trajectory pattern, called T-pattern, is a sequence of triples:

$$S = \{(x_0, y_0, t_0) \dots (x_i, y_i, t_i) \dots (x_n, y_n, t_n)\} \quad (1)$$

where t_i is a time annotation, $\forall_{0 \leqslant i < n}$, $t_i < t_{i+1}$, and (x_i, y_i) is a 2-dimensional point.

Definition 1 is beyond the concept of temporally annotation sequences (*TAS*)[24]. *TAS* is an extension of sequential patterns with the transition times between its elements. An illustrative example of *TAS* over the railway travel routes along the regions of interest in China denoted by the latitude and longitude positions is given as[5]: $(126.7, 45.8) \xrightarrow{11.5} (116.4, 39.9) \xrightarrow{26} (104.1, 30.7)$, representing a sequence that starts from the city of *Harbin*, then after 11.5 hours reaches *Beijing* railway interchange, and finally arrives at the tourist destination *Chengdu* after 26 hours trip. Note that we use trajectory pattern and trajectory sequence interchangeably.

The important concept of T-pattern mining is the frequency based on the *support count*, which is the number of input sequences that contain the specified *TAS*. The notion of τ-containment[10] with a time constrain τ is defined as follows.

Definition 2. (τ-containment) Given a time tolerance τ, a *TAS* $T_1 = (s_1, \tau_1)$ the length of which is m, and the other *TAS* $T_2 = (s_2, \tau_2)$ the length of which is n with $m \leqslant n$, where s_1 and s_2 are T-patterns. T_1 is τ-contained in T_2, denoted as $T_1 \preceq_\tau T_2$, iff there exist a serial of integers $0 \leqslant i_0 < \ldots < i_n \leqslant m$ satisfying that:

1. $\forall_{0 \leqslant k \leqslant n} \quad s_{1,k} \subseteq s_{2,i_k}$
2. $\forall_{k \leqslant 1 \leqslant n} \quad |\alpha_{1,k} - \alpha_{*,k}| \leqslant \tau$

where $\forall_{1 \leqslant k \leqslant n} \quad \alpha_{*,k} = \Sigma_{i_{k-1} < j \leqslant i_k} \alpha_{2,j}$.

We say that T_1 is τ-constrained in T_2 if there is an occurrence of T_1 in T_2 (Condition 1) having transition time similar to the temporal annotations in T_1 (Condition 2). To facilitate understanding, we give an example of τ-containment as follows.

1. $\{i_1\} \xrightarrow{5} \{i_2, i_3\} \xrightarrow{7} \{i_4, i_5\}$
2. $\{i_1\} \xrightarrow{4} \{i_2, i_3, i_4\} \xrightarrow{3} \{i_6, i_7\} \xrightarrow{6} \{i_4, i_5, i_8\}$

where the itemset in T_1 occurs in T_2, and the transition time of the occurrence differs at most by 2 time units (i.e., 3+6-7=2). Therefore, if $\tau \geqslant 2$, we conclude that $T_1 \preceq_\tau T_2$.

Definition 3. (Sub-trajecotry) A trajectory sequence $\alpha = (\alpha_1 \xrightarrow{\tau_1} \alpha_2 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_{n-1}} \alpha_n)$, where α_i is the ith itemset consisting of multiple 2-dimensional points represented by (x_i, y_i) and denoted as $\alpha_i = \{p_1, p_2, \ldots, p_n\}$. $\tau_1, \ldots, \tau_{n-1}$ are temporal annotations. α is a sub-trajectory of another trajectory sequence $\beta = (\beta_1 \xrightarrow{\tau_1} \beta_2 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_{m-1}} \beta_m)$, where β_j is the jth itemset and $\beta_j = \{q_1, q_2, \ldots, q_m\}$, denoted as $\alpha \sqsubseteq \beta$, if there exist integers $i \leqslant j$ such that $\alpha_i \subseteq \beta_j$ for all α_i.

Note that the relationship of $\alpha_i \subseteq \beta_i$ implies that the points in α_i appear at least once in itemset β_i.

By Definitions 2 and 3, the support and parallel mining of frequent trajectory patterns can be straightforwardly defined by extending the concept of τ-support[10].

The support count in parallel computing is distinct from that is based on a single processor. In general, the support for a sequence is defined as the fraction of the total data-sequences that contain this sequence[12]. Whereas, there exist lots of processors in a distributed environment, the support w.r.t. the ith processor is denoted as P_i. Assume that there is a trajectory pattern T_1, the number of T_1 appearing at the processor P_i is called the local support count, denoted as $\mathscr{L}Count_i(T_1)$. By extending **support count** and **τ-containment**, the formal concept of the local τ-support is defined below.

Definition 4. (Local τ-Support) Let P = $\{p_1, p_2, \ldots, p_m\}$ be a set of processors, Φ_i be the set of trajectory patterns that is assigned to $p_i (i = 1, 2, \ldots, m)$, τ be a time tolerance, and $minSupport \in [0, 1]$ be the minimum support threshold. The local τ-support is defined as[10]:

$$\tau\text{-}\mathscr{L}Supp_i(T) = \frac{|T \preceq_\tau T^* \& T^* \in \Phi_i|}{|\Phi_i|} \qquad (2)$$

where i is the serial number of processors and $i \in$1,2,...,m. T is frequent at Φ_i if τ-$\mathscr{L}Supp_i(T) \geqslant minSupport$.

Apparently, $\mathscr{L}Count_i(T)$=τ-$\mathscr{L}Supp(T) \times |\Phi_i|$ from Equation 2. The global support count $\mathscr{G}Count(T)$ corresponding to trajectory sequence T is calculated by Equation 3 as below. It collects the local support information from processors to determine whether a trajectory sequence is frequent.

$$\mathscr{G}Count(T) = \Sigma_{0 \leqslant i \leqslant m} \quad \mathscr{L}Count_i(T) \qquad (3)$$

Given a T-pattern T and a minimum support count *min_count*, if $\mathscr{G}Count(T) \geqslant min\_count$, we say that T is globally frequent. T is called a global FT-pattern.

Based on the above concepts, a general definition of the parallel FT-pattern mining[6] can be defined as follows.

Definition 5. (Parallel FT-pattern mining) Given a trajectory database D, a set of processors P, a time tolerance τ and a minimum support threshold *min_supp*, the parallel frequent trajectory pattern mining aims to find all global FT-patterns s satisfying that:

$$\mathscr{G}Supp_{D,P,\tau}(s) \geqslant min\_supp$$

where $\mathscr{G}Supp_{D,P,\tau} = \frac{\mathscr{G}Count(s)}{|D|}$ is the global support of the trajectory s at P, and s is the input trajectory $S \in D$ satisfying $s \preceq_\tau S$.

The T-pattern mining problem is complex in the parallel computing environment, since the communication overhead across multiple processors is costly in general cases for frequent sending and responding to requests in a parallel communication network. In particular, an efficient and effective parallel T-pattern mining algorithm should contain the following characteristics[22].

- *The approach must be more accurate and efficient than that of a single processor with similar parameter settings*. The T-patterns discovered by multiple processors should be consistent with the data mining results from a single processor. The method does not only minimize I/O cost by reducing the database scans, but also minimizing the computation cost by developing efficient search schemas.
- *Parallel algorithm should have low communication overhead*. For parallel computing, the communication overhead is costly if the processor frequently sends and responses to requests in a communication network composed of multiple processors. Accordingly, it is essential to design a more efficient parallel algorithm to reduce the communication cost across processors.
- *The subprocedures of a parallel algorithm must be executed asynchronously*. Each processor should work separately without any need for sharing or synchronization.

Before introducing our approach, we have to illustrate the properties from the parallel T-pattern mining problem. In the following section, we will discuss the properties of frequent trajectory patterns and provide the theoretical foundations in parallel T-pattern mining.

4. Property Analysis of Parallel T-patterns

In this section, we analyze the properties of T-patterns and introduce the methodology of T-patterns mining in a parallel computing environment. Firstly, we will introduce some useful lemmas as follows[6].

Lemma 1. *For a global FT-pattern T, there exists at least one processor p_i such that T and its sub-trajectories are globally frequent at p_i*[23].
Proof: We can prove this lemma by the method of reduction to absurdity. Suppose there exists no such processor, in terms of parallel sequential pattern mining, we know that $\mathscr{L}Count_i(T) < min\_count_i (i = 1,2,\ldots,m)$. Therefore, the sum of the number of T in the trajectory database should satisfy:

$$\begin{aligned} \because \mathscr{G}Count(T) &= \mathscr{L}Count_1(T) + \ldots + \mathscr{L}Count_m(T) \\ &< min\_count_1 + \ldots + min\_count_m \\ &= min\_supp \times \{|d_1| + \ldots + |d_m|\} \\ &= min\_supp \times |D| \end{aligned}$$

$\therefore \mathscr{G}Supp(T) \ngeq min\_supp$
$\therefore T$ is not a global FT-pattern, which is a contradiction. Therefore, the assumption does not hold. In the way, T is regarded as a global FT-pattern.

All sub-trajectories of T are global FT-patterns at p_i based on the **Apriori** property. □

The commonly-used approach for mining sequential patterns is the tree projection algorithm[14,15,25,26]. The key idea behind this category approaches is to construct a projection tree. In general, the tree is in lexicographical order and each node is associated with a *k*-itemset. In this paper, we employed the PrefixSpan tree projection technique[15] to construct a T-pattern tree. Here, we give a formal definition of FT-pattern tree as below.

Definition 6. (FT-pattern tree)[6,22] A FT-pattern tree (FTP-tree) contains all the frequent trajectory patterns. A trajectory sequence starting from the root node to a node at the *k*th level is called a L_k-pattern. The predecessor above the *k*th level node is its prefix, its corresponding T-pattern is called a L_{k-1}-pattern, and its length is *k*-1. The child node below the *k*th level node is its suffix, its corresponding T-pattern is called a L_{k+1}-pattern. FTP-tree can be partitioned into multiple subtrees by L_k-patterns, and the corresponding subtree is called a L_k-subtree at the *k*th level.

Definition 7. (Local subtree, Global subtree)[22] The subtree composed of local FT-patterns is called local subtree($\mathscr{L}$Subtree). Similarly, the global subtree is

such tree that is consisted of the global FT-patterns, denoted as $\mathscr{G}$Subtree. For any global L_k-pattern γ, its corresponding $\mathscr{G}$Subtree, which treats its k+1-level node as its root node, is called γ-Subtree. If T is a global T-pattern at p_i, then the local subtree that contains T at p_i is called T-$Subtree_i$, and the corresponding projection database is called T-DB_i.

Lemma 2. *If a T-pattern is globally frequent, it is locally frequent at each processor.*
Proof: Given a global FT-pattern γ, a set of processors $\{p_1,p_2,\ldots,p_m\}$, a trajectory database $|D| = |d_1|+|d_2|+\ldots+|d_m|$.

$$\begin{aligned}
\because \mathscr{G}Count(\gamma) &= \mathscr{L}Count_1(\gamma)+\ldots+\mathscr{L}Count_m(\gamma)\\
&\geqslant min_supp\times|D|\\
&= min_supp\times\{|d_1|+\ldots+|d_m|\}\\
&= min_count_1+\ldots+min_count_m\\
\because min_count_i &> 0, \forall i\in(1,2,\ldots,m)\\
\therefore \mathscr{G}Count(\gamma) &> min_count_i.
\end{aligned}$$

therefore, γ is locally frequent at each processor.

Theorem 3. *The set of global FT-patterns is a subset of local L_k-subtrees*[22].
Proof: Given a global FT-pattern γ, by Lemma 2, we can see that γ is a local FT-pattern, which implies that γ at least appears in one L_k-subtree and locally frequent. However, it cannot guarantee that γ is frequent in each local L_k-subtree, which means γ occurs in this L_k-subtree but infrequent. Therefore, the theorem is proved. □

5. Parallel T-pattern Mining Algorithm

In this section, we propose a new parallel trajectory pattern mining algorithm, called **plute**. It contains the following essential techniques: (1) use the PrefixSpan projection approach to decompose the search space of sequential patterns in order to reduce the candidate subsequences, (2) employ a new parallel formulation approach that integrates google's MapReduce model[9] to distribute the data and the mining tasks among the available processors over the prefix sequential patterns, (3) further prune the candidate T-patterns by utilizing the properties of local FT-patterns and global FT-patterns, and finally (4) use an asynchronous algorithm to tune the subprocedures of parallel computations in order to obtain the global FT-patterns.

5.1. PrefixSpan Algorithm

PrefixSpan is an efficient sequential pattern mining algorithm. To the best of our knowledge, Giannotti et al. was the first to extend PrefixSpan to discover the frequent T-patterns[10]. The key idea behind this approach is as[15]: for each frequent item a, a projection of this initial database D is created, denoted as $D|_a$, and a mining process contains: (i) finding frequent sequential patterns only containing item a, (ii) finding frequent trajectory patterns containing other items (e.g., b), but no item after them, and (iii) finding other subsets of FT-patterns in the similar manner. The main idea of this approach is that any sequence starting with a can be obtained by only analyzing $D|_a$, which can help reduce the candidate items. Then, a frequent pattern ab (or (ab)) is derived from item b that is frequent in $D|_a$, and a new smaller projection database $D|_{ab}$ (or $D|_{(ab)}$) is recursively calculated for finding longer frequent patterns starting with ab (or (ab)).

The T-pattern mining problem is distinct from traditional sequential pattern mining algorithms, since *TAS* are constrained by the time tolerance τ. So, we extended the definition of T-sequence[10] by combining the parallel information.

Definition 8. (Parallel trajectory sequence) Given a projected, time-stamped trajectory sequence $S = \{(s_1,t_1),(s_2,t_2),\ldots,(s_n,t_n)\}$, obtained as projection of sequence S_0 w.r.t. the prefix s^* (i.e., $S = S_0|_{s^*}$), and a set of processors q_i, where i is the serial number of processors. Parallel trajectory sequence (PT-sequence) is defined as the couple (S,N_i), where $N_i = \{(a_{i_1},p_{i_1}),(a_{i_2},p_{i_2})\ldots,(a_{i_n},p_{i_m})\}$ at q_i: each couple (a_{i_j},p_{i_j}) represents one occurrence of the prefix s^* in the original sequence S_0, a_{i_j} is the sequence of time-stamps of such an occurrence, and p_{i_j} is a pointer to the element of S where the occurrence terminates or the symbol $\emptyset$ if such element is not in S.

PT-sequence explicitly integrates such information in a trajectory together with the time point in

the sequence and the processor where the sequence assigned to. To facilitate understanding, we give an illustrative example as follows.

Example 1. Given a time annotated trajectory sequence at Processor 1, $S_1 = \{(\{a\},1),(\{a,b\},2),(\{c\},3),(\{a,c\},4)\}$, the PT-sequence w.r.t. prefix a is the couple $(S_1|_a, N_1)$, where:

$$S_1|_a = \{(\{a,b\},2),(\{c\},3),(\{a,c\},4)\}$$
$$N_1 = \{(\langle 1\rangle,\emptyset),(\langle 2\rangle,\rightarrow 2),(\langle 3\rangle,\emptyset),(\langle 4\rangle,\rightarrow 4)\}$$

We use the similar T-pattern projection approach proposed in Ref. 10. As shown in Example 1, we first project the T-sequence w.r.t. the prefix 'a', then perform an enlargement projection w.r.t. 'b' until the last annotation cannot be enlarged.

Here, the notation $\rightarrow 2$ stands for "*pointer to element having time = 2*". The first occurrence of 'a' is moved into the prefix, so its corresponding pointer is set to $\emptyset$. In addition, since the third element in N_1 does not treat 'a' as its prefix, its pointer is set to $\emptyset$ as well. Then, we consider the case w.r.t. the prefix 'ab' beyond $S_1|_a$, we can obtain:

$$S_1|_{ab} = \{(\{c\},3),(\{a,c\},4)\}$$
$$N_1 = \{(\langle 1,2\rangle,\emptyset),(\langle 2,3\rangle,\rightarrow 3),(\langle 2,4\rangle,\rightarrow 4)\}$$

Note that N_1 has two time annotations for each occurrence of the prefix 'ab', since the prefix has two items distributed in two itemsets.

5.2. Parallel Formulation

The overall structure of the computations performed by the FTP-tree projection for discovering FT-patterns is generated by the PrefixSpan algorithm. Generally, there are two methods that can be used to decompose the computations[27]. The first approach is the data parallel formulation that exists in computing the support at each node, whereas the second method exploits the task formulation that lies in the tree-based nature of the computation. We integrate the basic idea of FDMSP[22] and MapReduce[9] to perform the data parallel and task parallel formulations, respectively, in order to maximize the parallel processing of computations.

First, we use the data parallel formulation to decompose the computations associated with counting the support of each T-pattern in a projection tree. The formulation works as follows.

The trajectory database is initially partitioned into k parts of equal size and each one is assigned to a distinct processor, where k is the serial number of a processor. Then we use the following steps to generate the global L_1-patterns. Firstly, compute the union of the local L_1-pattern at each processor. Secondly, compute the support count (density) of each local L_1-patterns based on the *ComputeDensity* approach[3]. Thirdly, each processor broadcasts the support counts of its T-patterns to any other processor. Finally, each processor sum up the support count of every local T-pattern γ. If the value of $Count(\gamma) \geqslant min\_count$, γ is treated as a global L_1-pattern, then output it into a global L_1-pattern set $\mathscr{S}$. The communication complexity of computing the $\mathscr{G}Count$ of each T-pattern is $O(k^2)$. The detail of the data parallel formulation is available in Ref. 6.

The second phase of the parallel processing is the task parallel formulation, i.e., distributing the tasks among processors. In this study, we borrow the basic idea behind the MapReduce programming model[9] to achieve the task parallelism.

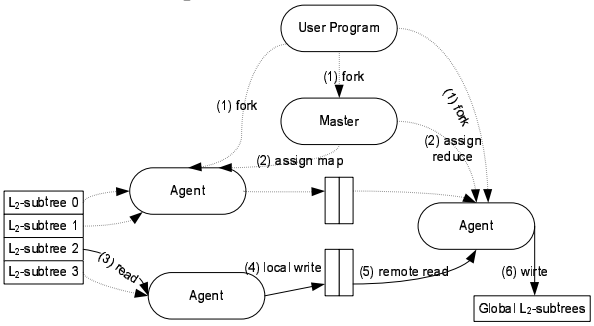

Fig. 1. Workflow overview.

Figure 1 shows the workflow of a MapReduce based task parallel formulation. When the user calls the MapReduce function, the following series of operations occur. Note the numbered labels in Figure 1 correspond to the numbers in the list below[9].

1. The MapReduce library in the user program partitions the input L_1-subtree into M pieces of L_2-subtrees. Then, it starts up many copies of the program on a cluster of processors.

2. The master processor is special, and it is responsible for assigning work to the rest

agents. There are M map tasks and R reduce tasks to distribute. The master picks idle agents and distributes each one a map task or a reduce task.

3. A agent who is distributed a map task reads the L_2-subtree of the corresponding input split. It parses the key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered into memory.

4. The buffered pairs are written to the local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for transferring these locations to the reduce agents.

5. When a reduce agent is activated by the master about these locations, it uses remote procedure calls to read the buffered L_2-subtrees from the map agents. When a reduce agent has read all intermediate L_2-subtrees, it merges them into one single L_2-subtree so that all occurrences of the similar prefix are grouped together. The detail of the merge approach is given in Example 2.

6. After the reduce agent receives the counting value from the map agents, it will find the FT-patterns as well as output the L_2-patterns to $\mathscr{G}$. In this phase, the reduce agent iterates over the intermediate L_2-subtrees and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function.

7. Finally, the agents will generate the global L_2-subtrees by pruning infrequent T-patterns, and the master wakes up the user program. The MapReduce call in the user program returns back to the user code and iterates the above steps to achieve the task parallel formulation of generating the global L_k-subtrees ($k \geqslant 1$).

Example 2. Given three processors p_1, p_2, p_3, the FT-patterns w.r.t. prefix $\{$a,b$\}$ at each processor are as follows:

$L_1|_{ab} = \{\{ab\},\{(ac)\},\{(d)\},\{a(c)f\}\}$
$L_2|_{ab} = \{\{ab\},\{a(cd)\},\{(d)\},\{b(de)\},\{(cd)\},\{(f),\{df\}\}$
$L_3|_{ab} = \{\{(a)\},\{(ab)\},\{(c)\},\{(d)\},\{(cd)\},\{f\},\{(f)\}\}$

In Figure 2, the subtrees corresponding to the above FT-patterns and the union tree clustered to p_2 below the above three subtrees.

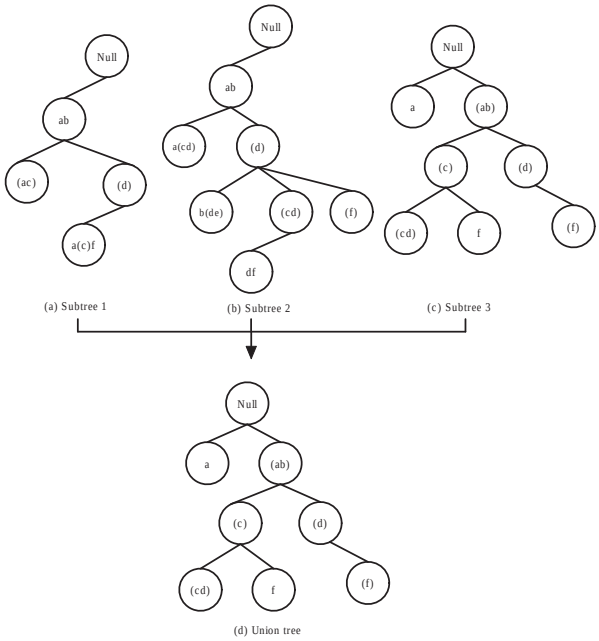

Fig. 2. Subtree union operation.

5.3. Candidate T-sequence Pruning

The pruning phase in the plute algorithm plays an essential role in the improvement of the mining efficiency. In order to save the storage space of candidate T-patterns, we combine the pruning strategy[23] and the annotation-based projection pruning approach[10].

The basic idea of pruning the annotation-based projection[10] is to determine whether the time annotation within its hyper-cubic neighborhood (the hyper-rectangle centered in each dataset point and having the edge of 2τ, where τ is the time tolerance) is dense, which means any annotation within the specified interval is frequent. Since a T-pattern that does not contain any useful occurrence of the

prefix can generate a large volume of useless sequences, if no annotation within the neighborhood is frequent, the item could safely be removed. The detailed pruning strategies is given in Ref. 23.

The sequence extension can be divided into two categories: (i) item extension $\langle t_1 t_2 \dots t_n(p) \rangle$, where p is an item, and (ii) sequence extension $\langle t_1 t_2 \dots t_n p \rangle$, where p is a suffix of the original sequence. Here, the sequence extension is called a superset of the initial T-pattern. Based on the **Apriori** property[28], we can straightforwardly obtain the following two corollaries[23].

Corollary 4. *The superset of a local infrequent T-pattern is infrequent.*

Corollary 5. *The superset of a global infrequent T-patterns is infrequent.*

Based on the above corollaries, plute adopts the following two pruning strategies[23].

1. *Item pruning*. Given two extension T-sequences $S \oplus \mu_1 \oplus \mu_2 \oplus \dots \oplus \mu_n$ and $S \oplus_i \nu$, where ν is an item extension, denoted as $\oplus_i$. If $S \oplus_i \nu$ is not a global FT-pattern, $S \oplus \mu_1 \oplus \mu_2 \oplus \dots \oplus \mu_n \oplus_i \nu$ is not globally frequent, which means the possible item extension of $S \oplus \mu_1$ to the last item ν can be disregarded.

2. *Sequence pruning*. Given two extension T-sequences $S \oplus \mu_1 \oplus \mu_2 \oplus \dots \oplus \mu_n$ and $S \oplus_s \nu$, where ν is an sequence extension, denoted as $\oplus_s$. If $S \oplus_s \nu$ is not a global FT-pattern, $S \oplus \mu_1 \oplus \mu_2 \oplus \dots \oplus \mu_n \oplus_s \nu$ is not globally frequent, which means the possible item extension of $S \oplus \mu_1$ to ν should be pruned.

Since there exist several T-sequences like $S \oplus_i \nu$ and $S \oplus_s \nu$ as the sequence is augmented. The candidate pruning can help eliminate the unnecessary T-patterns in order to reduce the communication cost before sending the support computation request to other processors.

6. Experimental Evaluation

6.1. Experimental setup

In this section, we report the experimental studies by comparing plute with the typical parallel sequential pattern mining algorithm (PartSpan)[6] which performs the task parallel formulation by using the hash partitioned sequential pattern mining approach HPSPM[19]. The experimentations consist of measuring two important parameters including: execution time and communication cost. Without loss of generality, all algorithms were run on the real as well as the synthetic data sets, respectively. The description of the data sets are given as follows.

1. Trucks data set consists of 276 trajectories from 50 trucks delivering concrete to several construction places around Athens metropolitan area in Greece for 33 days, for a total of 112,203 points.[§]

2. The synthetic data are generated by Brinkhoff's network-based generator of moving objects[29]. It contains 100,000 trajectories of one day movement over the road-network of Oldenburg. The data size is about 225Mb.

All experiments are conducted on a computer workshop consisting of 8 PC with Pentium IV 2.4 GHz CPU, 512 Mb of RAM, and running Microsoft Windows XP Professional Operating System.

6.2. Comparison of Parallel Time

In this series of experimentations, we will evaluate the parallel execution time under a variety of parameter settings. We first compare the parallel execution time of plute with PartSpan by changing the number of processors from 4 to 8. Figures 3 and 4 illustrates the execution time between these two algorithms in the real as well as the synthetic data sets at a minimum support of 0.1%.

[§] Available from http://www.rtreeportal.org/

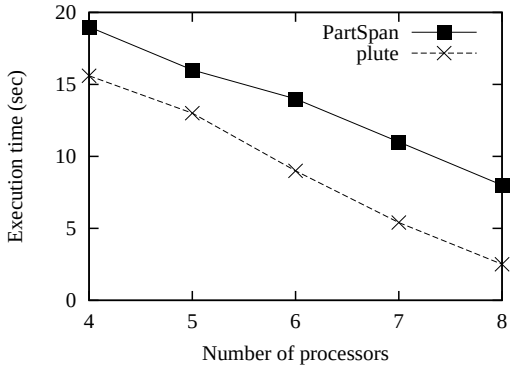

Fig. 3. Real data

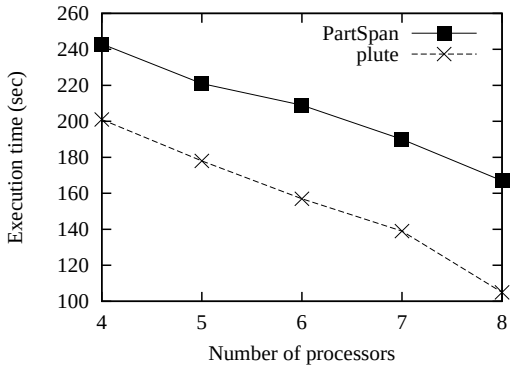

Fig. 4. Synthetic data

As we can see from Figures 3 and 4, the runtime of PartSpan and plute decreases as the number of processors increases. This is due to the specific data and task parallel formulation approach used in these two algorithms. In particular, plute performs better than PartSpan in all cases with distinct processors, and reduces the execution time with respect to PartSpan by a factor up to 1.85 in the real data set and 1.35 in the synthetic data set, respectively. This is because plute employs MapReduce programming model to find the global T-patterns which helps reduce the communication cost, and we will further explore the communication cost in Section 6.3. In addition, it specifies a *Map* and *Reduce* function to schedule the program's execution across a series of processors and can handle machine failures. Whereas, PartSpan only uses the hash partitioned sequential pattern mining approach[19] to assign tasks to processors.

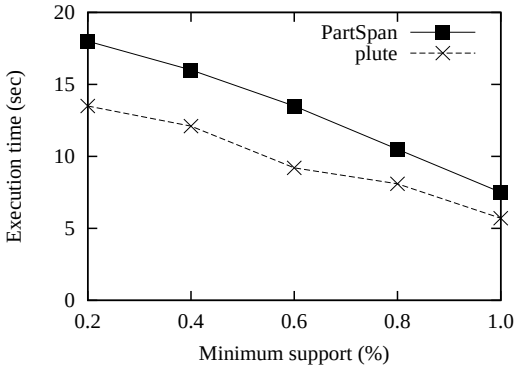

Fig. 5. Real data

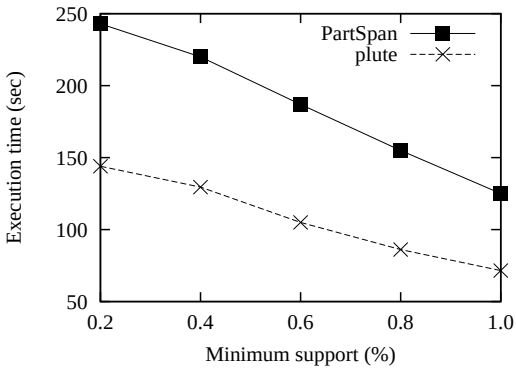

Fig. 6. Synthetic data

Figures 5 and 6 show the execution time comparison of these two algorithms as the minimum support threshold is increased from 0.2% to 1% in the parallel computing environment of four processors in the real and the synthetic data sets, respectively. We can see that the execution time of PartSpan and plute decreases linearly with the minimum support. This is duo to the specific candidate pruning strategies, which reduce the candidate trajectory patterns in a nearly linear fashion with the minimum support that have been addressed in Ref. 21. We also find that plute is the winner and achieves the biggest gap with regard to PartSpan of 1.47 times in the real data set and 1.8 times in the synthetic data set with the minimum support ranging from 0.2% to 1%. The reason is that the MapReduce based task parallel formulation approach applies a number of optimizations that are targeted at reducing the amount of trajectory patterns sent in the network, e.g., the locality optimization allows us to read trajectory patterns from local disks[9].

Finally, we study the effect of the changing data size on the parallel performance among these algorithms with four processors. Note that the experimental result are similar in any number of processors. We can see how PartSpan scales up as the cardinality of data is increased ten-fold, from 500k to 5M in the real data set (in Figure 7)and from 20M to 200M in the synthetic data set (in Figure 8), respectively. The minimal support is set to 0.5% in this set of experiments.

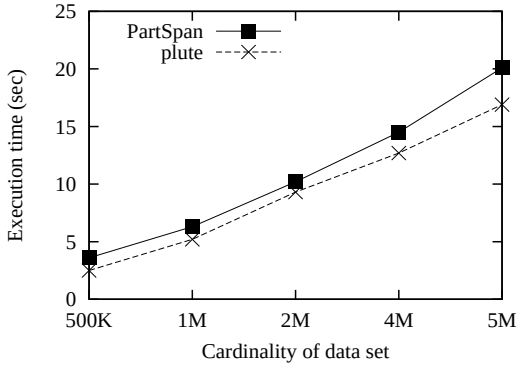

Fig. 7. Real data

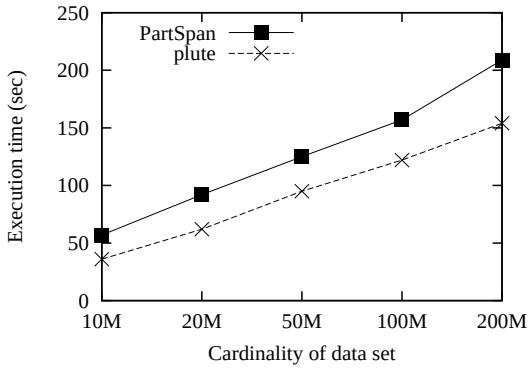

Fig. 8. Synthetic data

Figures 7 and 8 show that the execution time w.r.t. these two algorithm increases as the data size grows. The execution time of PartSpan and plute increase linearly with respect to the varying data sets, which implies that these two algorithm are more scalable. Because PartSpan and plute apply the prefix-based item and sequence pruning strategies that are not sensitive to the changing cardinality of data.

6.3. Communication Cost Comparison

In parallel systems, the communication cost across distinct processors is often high, thus we have to further analysis the effect of this evaluator. In this set of experiments, the communication cost includes the broadcasting time and the respond time to other processors. We observe the communication time between PartSpan and plute executed at four processors as the cardinality of the synthetic data sets increases from 10M to 200M at the minimum support of 0.2%.

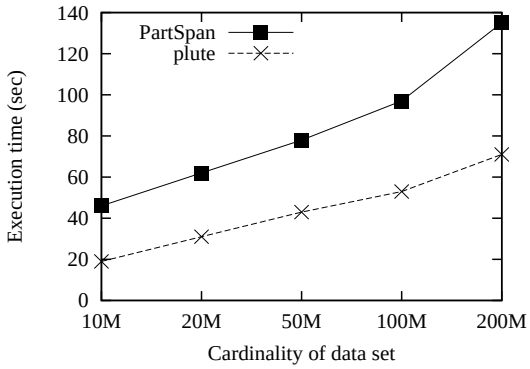

Fig. 9. (Communication time comparison: synthetic data

The results are shown in Figure 9, and they agree with the cases in the real data sets as well. By Figure 9, we see that the communication cost in terms of plute increases linearly with respect to the cardinality of data. The plute algorithm is the winner in each experiment. This is because plute employs the MapReduce model to discover the global T-patterns, each processor only needs to sent the L_k-subtree (where k is the level of subtrees) to its corresponding agent instead of broadcasting to all processors that helps reduce the communication cost.

7. Conclusions and Future Directions

In this paper, we propose a novel parallel sequential time annotated patterns mining method for massive trajectory data, called plute. Its general idea is to partition the search space by the prefix-projection approach, and introduce the parallel strategy to divide the parallel computation into the data formulation and the MapReduce based task formulation. To further improve the mining efficiency, two specific

candidate strategies are applied, i.e., item pruning and sequence pruning. The performance study describes in detail the merits of plute with regard to different parameter settings in the real as well as the synthetic data sets.

The plute algorithm is a new methodology for efficiently mining the trajectory patterns in massive data, it can also be directly applied to mining other sequential patterns with time annotations. Our future research direction includes: (i) extend plute to mining the web logs; (ii) optimize the low-level parameters, e.g., the support for parallel computations; (iii) design a prediction method of uncertainty trajectories in moving databases based on plute; (iv) apply other data mining approaches[30,31,32,33] to improve the performance of plute.

References

1. H. M. O. Mokhtar and J. Su, "Universal trajectory queries for moving object databases," *MDM'04: Proc. IEEE Intl. Conf. on Mobile Data Management*, 133–144 (2004).
2. R. Agrawal and R. Srikant, "Mining sequential patterns," *Proc. Intl. Conf. on Data Eng.*, 3–14 (1995).
3. F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, "Trajectory pattern mining," *Proc. ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, 330–339 (2007).
4. F. Giannotti, M. Nanni, D. Pedreschi, and F. Pinelli, "Mining sequences with temporal annotations," *Proc. ACM Symposium on Applied Computing*, 593–597 (2006).
5. S. Qiao, C. Tang, H. Jin, T. Long, S. Dai, Y. Ku, and M. Chau, "Putmode: prediction of uncertain trajectories in moving objects databases," *Appl. Intell.*, DOI: 10.1007/s10489-009-0173-z (2009).
6. S. Qiao, C. Tang, S. Dai, M. Zhu, J. Peng, H. Li, and Y. Ku, "PartSpan: parallel sequence mining of trajectory patterns," *Proc. Intl. Conf. on Fuzzy Systems and Knowledge Discovery*, **V**, 363–367 (2008).
7. K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley (2006).
8. M. J. Zaki, "Parallel sequence mining on shared-memory machines," *J. of Parallel and Distrib. Computing*, **61(3)**, 401–426 (2001).
9. J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. of the ACM*, **51(1)**, 107–113 (2008).
10. F. Giannotti, M. Nanni, and D. Pedreschi, "Efficient mining of temporally annotated sequences," *Proc. SIAM Conf. on Data Mining*, 346–357 (2006).
11. T. G. Dietterich and R. S. Michalski, "Discovering patterns in sequences of events," *Artif. Intell.*, **25(2)**, 187–232 (1985).
12. R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," *Proc. Intl. Conf. on Extending Database Technology*, 3–17 (1996).
13. M. J. Zaki, "Efficient enumeration of frequent sequences," *Proc. Intl. Conf. on Information and Knowledge Management*, 68–75 (1998).
14. J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. C. Hsu, "Freespan: frequent pattern-projected sequential pattern mining," *Proc. ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, 355–359 (2000).
15. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. C. Hsu, "Prefixspan: Mining sequential patterns by prefix-projected growth," *Proc. Intl. Conf. on Data Engineering*, 215–224 (2001).
16. J. C. Augusto, J. Liu, P. McCullagh, H. Wang, and J. Yang, "Management of Uncertainty and Spatio-Temporal Aspects for Monitoring and Diagnosis in a Smart Home" *Intl. J. of Comput. Intell. Syst.*, **1(4)**, 361–378 (2008).
17. H. Cao, N. Mamoulis, and D. W. Cheung, "Mining frequent spatio-temporal sequential patterns," *Proc. IEEE Intl. Conf. on Data Mining*, 82–89 (2005).
18. J. Lee, J. Han, and K. Whang, "Trajectory clustering: a partition-and-group framework," *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 593–604 (2007).
19. T. Shintani and M. Kitsuregawa, "Mining algorithms for sequential patterns in parallel: Hash based approach," *Proc. Pacific-Asia Conf. on Research and Development in Knowledge Discovery and Data Mining*, 283–294 (1998).
20. S. Cong, J. Han, and D. Padua, "Parallel mining of closed sequential patterns," *Proc. ACM SIGKDD Intl. Conf. on Knowledge Discovery in Data Mining*, 562–567 (2005).

21. V. Guralnik and G. Karypis, “Parallel tree-projection-based sequence mining algorithms,” *Parallel Comput.*, **30(4)**, 443–472 (2004).
22. X. Zou, W. Zhang, Y. Liu, and Q. Cai, “Study on distributed sequential pattern discovery algorithm,” *J. of Software*, **16(7)**, 1262–1269 (2005).
23. Z. Gong, K. Hu, L. Da, and C. Zhang, ”DMGSP: an algorithm of distributed mining global sequential pattern on distributed system,” *J. of Southeast Univeristy (Natural Science Edition)*, **37(4)**, 574–579 (2007).
24. F. Giannotti, M. Nanni, D. Pedreschi, and F. Pinelli, “Mining sequences with temporal annotations,” *Proc. ACM Symposium on Applied Computing*, 593–597 (2006).
25. R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad, “A tree projection algorithm for generation of frequent item sets,” *J. of Parallel and Distrib. Computing*, **61(3)**, 350–371 (2001).
26. V. Guralnik, N. Garg, and G. Karypis, “Parallel tree projection algorithm for sequence mining,” *Proc. Intl. Euro-Par Conf. Manchester on Parallel Processing*, 310–320 (2001).
27. V. Kumar, A. Grama, A. Gupta, and G. Karypis, “Introduction to parallel computing: design and analysis of algorithms,” Benjamin-Cummings Publishing Co., Inc., Redwood City (1994).
28. R. Agrawal, T. Imielinski, and A. N. Swami, “Mining association rules between sets of items in large databases,” *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 207–216 (1993).
29. T. Brinkhoff, “Generating traffic data,” *IEEE Data Engineering Bulletin*, **26(2)**, 19–25 (2003).
30. S. Qiao, C. Tang, H. Jin, J. Peng, D. Davis, and N. Han, “KISTCM: knowledge discovery system for traditional Chinese medicine,” *Appl. Intell.*, **32(3)**, 346–363 (2010).
31. X. Li, D. Ruan, J. Liu, and Y. Xu, “A linguistic-valued weighted aggregation operator to multiple attribute group decision making with quantative and qualitative information” *Intl. J. of Comput. Intell. Syst.*, **1(3)**, 274–284 (2008).
32. S. Qiao, C. Tang, H. Jin, S. Dai, X. Chen, M. Chau, and J. Hu, “Processing constrained k-Closest pairs queries in crime databases,” *Annals of Inform. Syst.*, **9**, 59–76 (2009).
33. G. Zhang, C. Shi, and J. Lu “An extended KTH-best approach for referential-uncooperative bilevel multi-follower decision making” *Intl. J. of Comput. Intell. Syst.*, **1(3)**, 205–214 (2008).