PartSpan: Parallel Sequence Mining of Trajectory Patterns

Shaojie Qiao†‡, Changjie Tang†, Shucheng Dai†, Mingfang Zhu†
Jing Peng†, Hongjun Li†, Yungchang Ku∗

†School of Computer Science, Sichuan University, Chengdu 610065, China
{qiaoshaojie, tangchangjie}@cs.scu.edu.cn
‡School of Computing, National University of Singapore, Singapore 117590, Singapore
∗Department of Information Management, Yuan Ze University, Taiwan 320, R.O.C

Abstract

The trajectory pattern mining problem has recently attracted increasing attention. This paper precisely addresses the parallel mining problem of trajectory patterns as well as the newly proposed concepts with regard to trajectory pattern mining. An efficient Parallel trajectory Sequential pattern mining (PartSpan) is proposed by incorporating three key techniques: prefix-projection, parallel formulation, and candidate pruning. The prefix-projection technique is used to decompose the search space as well as greatly reducing candidate trajectory sequences. The parallel formulation integrates the data parallel formulation and the task parallel formulation to partition the computations and to assign them to multiple processors in an efficient and effective manner that helps reduce the communication cost across processors. Representative experiments are used to evaluate the performance of PartSpan. The results show that PartSpan outperforms GSP-based and SPADE-based parallel algorithms in mining very large trajectory databases.

1 Introduction

Recently the popularity of wireless communication and location aware devices makes it possible for us to collect a large volume of trajectory data of moving objects. For instance, global positioning systems (GPS) are widely used to trace the locations of traffic facilities in an accurate and instantaneous manner. Through these tracking facilities, there accumulates an ever-increasing amount of trajectory data, which are time-stamped sets of events. Frequent trajectory pattern mining is a challenging problem and of great practical value especially for very large databases. The time annotated sequence is useful in a variety of areas, such as web log analysis [6], traffic tracking and analysis [7], animal movement extraction [12], and crime hotspot detection. For the above reasons, we should utilize the spatio-temporal information from trajectories to improve the computational efficiency, which provides an opportunity to automatically discover the useful knowledge from trajectory databases.

Parallel computing has recently become the dominant paradigm in computer science, mainly in the form of multicore processors [2]. In particular, it is suitable for the problem requiring a large volume of computations. Parallel computing is an alternative solution for sequence mining, which is called parallel sequence mining [16]. To take into account of the search space limitation, serial algorithms cannot provide scalability, in terms of the performance as well as the data size for very large databases [16]. The rapid development of multiprocessor systems provides opportunities for us to explore efficient parallel algorithms.

Aiming at solving the trajectory pattern (T-pattern) mining problem, we make the following contributions.

1. We accurately address the parallel trajectory pattern mining problem and introduce some new concepts with regard to trajectory parallel computing.
2. We propose a hybrid approach that combines the data parallel formulation and the task parallel formulation to decompose computations and to assign the computations of trajectory pattern mining to multiple processors in an efficient and economical manner with lower computation cost.
3. We proposed an efficient parallel mining algorithm of trajectory patterns, namely PartSpan, based on the prefix-projection and the parallel formulation.
2 Related Work

The sequential pattern mining has been widely investigated. An influential sequence mining algorithm is proposed by Agrawal et al. [1], called AprioriAll. However, the phase of transformation by substituting item sets in each transaction is costly. To better handle the problem of transforming in large databases, Srikant and Agrawal proposed an efficient sequential pattern mining approach, namely GSP [15]. GSP is a generalized algorithm aiming to find all frequent sequences without transforming databases.

An extension of sequential pattern mining is the spatio-temporal pattern mining problem. A popular approach is recently proposed by Giannotti et al. [7]. They addressed the trajectory pattern mining problem, in which the spatio-temporal pattern represents an aggregated extraction of several trajectories of moving objects within an observed area [7]. In particular, they also proposed a density-based algorithm for discovering regions of interest [7] by extending the work on the mining of TAS (temporally-annotated sequence). However, all the previous approaches for finding sequential patterns are serial algorithms. In case of a large sequence database, these algorithms require multiple passes over the database, which needs long calculation time. This motivates us to develop an efficient parallel approach that relies on multiple processors.

The parallel mining of sequential patterns has recently attracted much attention [14, 16, 4]. Zaki et al. [16] proposed a parallel algorithm for efficiently discovering sequential patterns in large databases, called pSPADE. An alternative method is the parallel tree projection algorithm [9, 10]. The recently proposed approaches include Zou’s FDMSP [17] and Gong’s DMGSP [8]. They use a similar strategy to compress local frequent sequential patterns into a lexicographic sequence tree in order to avoid translations of repeated prefixes.

To the best of our knowledge, there is no related work on parallel trajectory pattern mining. This problem is complex and specific, since the time-stamps of sequences are constrained with a specified time interval. Whereas, we have to take into account the minimum support threshold for eliminating infrequent sequential patterns.

3 Problem Statement

The important concept of trajectory pattern mining is the frequency based on support count, that is defined as the number of input sequences that contain the specified TAS [5]. In a distributed environment, there exists several processors, denoted as $p_i$. Given a trajectory pattern $T_1$, the number of $T_1$ occurring at processor $p_i$ is called the local support count, denoted as $LCount_i(T_1)$, where $i$ represents the $i$th processor. By extending the notion of $\tau$-containment [5], the concept of local $\tau$-support is defined as follows.

**Definition 1 (Local $\tau$-support)** Let $P = \{p_i|i = 1, 2, \ldots, m\}$ be a set of processors, $S_i$ be the set of trajectory patterns that is assigned to $p_i$, $\tau$ be a time tolerance, and $\text{min\_supp} \in [0, 1]$ be the minimum support threshold. The local $\tau$-support is defined as:

$$\tau-L\text{Supp}_p(Tr) = \frac{|Tr \preceq \tau, T^* \in S_i|}{|S_i|}$$

where $i$ is the processor serial number. $Tr$ is frequent if $\tau-L\text{Supp}_p(Tr) \geq \text{min\_supp}$.

Here, $Tr \preceq \tau, T^*$ means $Tr$ is contained in $T^*$ satisfying the time constraint [5]. It is apparent that $LCount_i(Tr) = \tau-L\text{Supp}_i(Tr) \times |S_i|$. The global support count $G\text{Count}(Tr)$ corresponding to $Tr$ is calculated by Equation 2 as follows.

$$G\text{Count}(Tr) = \sum_{0 \leq i \leq m} LCount_i(Tr)$$

Given a T-pattern $Tr$ and a minimum support $\text{min\_count}$, if $G\text{Count}(Tr) \geq \text{min\_count}$, we say that $Tr$ is globally frequent. $Tr$ is called a global FT-pattern. The formal definition of the parallel FT-pattern mining problem is shown below.

**Definition 2 (Parallel FT-pattern mining)** Given a trajectory database $D$, a set of processors $P$, a time tolerance $\tau$, and a minimum support threshold $\text{min\_supp}$, parallel frequent trajectory pattern mining problem is defined as finding global FT-patterns $l$ such that: $G\text{Supp}_{D,P,\tau}(l) \geq \text{min\_supp}$, where $G\text{Supp}_{D,P,\tau}(l) = \frac{G\text{Count}(l)}{|D|}$ is the global support of $l$, and $l$ is the input trajectory satisfying $l \preceq \tau, S$, where $S \in D$.

From the above concepts, we can obtain the following lemma.

**Lemma 1** For a global FT-pattern $\gamma$, there exist at least one processor $p_i$ such that $\gamma$ and its sub-trajectories are globally frequent at $p_i$.

**Proof:** This lemma is proved by reduction to absurdity. Suppose there exists no such processor, then we directly obtain that $\forall i \in [1, m], G\text{Count}_i(\gamma) < \text{min\_count}_i(i = 1, 2, \ldots, m)$. Thus, the sum of the number of $\gamma$ in the trajectory database satisfying:

$$\because G\text{Count}(\gamma) = \sum_{1 \leq i \leq m} L\text{Count}_i(\gamma) < \sum_{1 \leq i \leq m} \text{min\_count}_i$$

$$= \text{min\_supp} \times \sum_{1 \leq i \leq m} |d_i|$$

$$\therefore G\text{Supp}(\gamma) \not\geq \text{min\_supp}$$
\[ \gamma \text{ is not a global FT-pattern, which is a contradiction. The assumption does not hold, so } \gamma \text{ is a global FT-pattern.} \]

Based on the Apriori property, all sub-trajectories of \( \gamma \) are global FT-patterns at \( p_i \).

Here, we borrow the basic idea of Sequence Tree introduced in [17] and give a definition of FT-pattern tree used in the following section.

**Definition 3** (FT-pattern tree) A FT-pattern tree (FTP-tree) is composed of all frequent trajectory patterns. A trajectory sequence beginning from the root node to a node at the \( k \)-th level is called \( G_k \)-pattern. The ancestor above the \( k \)-th level node is its prefix, the corresponding FT-pattern is called \( G_{k-1} \)-pattern, and its length is \( k-1 \). The children node below the \( k \)-th level node is its suffix, and the corresponding FT-pattern is called \( G_{k+1} \)-pattern. FTP-tree can be partitioned into multiple subtrees with regard to distinct \( G_k \)-patterns, and the corresponding subtree that roots at the \( k \)-th level is called \( G_k \)-subtree.

In this paper, the subtree consisting of local FT-patterns is called local subtree [17], denoted as \( \mathcal{L} \text{Subtree} \). Similarly, the global subtree is such tree that is composed of global FT-patterns [17], denoted as \( \mathcal{G} \text{Subtree} \). For any global \( G_k \)-pattern \( \gamma \), its corresponding \( \mathcal{G} \text{Subtree} \), which treats the \( k \)-th level node as its root node, is denoted as \( \gamma \)-subtree.

### 4 PartSpan: Parallel T-pattern Mining

This section presents a new parallel trajectory pattern mining algorithm, called PartSpan (Parallel trajectory Sequential pattern mining). Its key idea is that: (1) use the PrefixSpan algorithm to decompose the search space of sequential patterns in order to reduce candidate subsequences, and (2) employ a new parallel formulation approach [9] to distribute trajectory data and the mining tasks across available processors. The detail of the PrefixSpan algorithm is reviewed in [13].

In general, there are two methods that can be used to decompose the computations [11], i.e., the data parallel formulation and the task formulation. We integrate both of these approaches and borrow the basic idea of FDMSP [17] to maximize the parallel processing of computations.

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

The original trajectory database is partitioned into \( k \) parts of equal size and each one is assigned to a distinct processor, where \( k \) is the number of processors. Then, we use the similar approach proposed in [17] to generate global \( G_1 \)-patterns: First, compute the union of local \( G_1 \)-patterns at each processor. Second, compute the support count (i.e., density) of each \( G_1 \)-pattern based on the \( \text{ComputeDensity} \) approach, introduced in [7]. Third, each processor broadcasts the support counts of its T-patterns to any other processor. Finally, each processor sums up the support count of every local T-pattern \( s \). If \( \text{Count}(s) \geq \text{min\_count} \), \( s \) is a global \( G_1 \)-pattern. Then, we output it into a global \( G_1 \)-pattern set \( S \). The data parallel formulation is shown as follows.

**Algorithm 1 Data Parallel Formulation**

| Input: a trajectory database \( \mathcal{D} \), a set of processors \( \mathcal{P} \), a minimum support threshold \( \delta \), a radius for spatial neighborhoods \( \epsilon \). |
| Output: a set of \( G_1 \)-patterns \( S \). |
| \( S = \text{null} \); |
| \( p_i \leftarrow \text{Distribute}(\mathcal{D}) \); |
| for \( i := 1 \) to \( n \) do |
| \( \gamma \) for each local T-pattern \( s \) in \( p_i \) do |
| \( C_s = \text{ComputeDensity}(s, \epsilon) \); |
| for \( k! = i \) do |
| \( p_k \leftarrow C_s \); |
| \( D_s = \Sigma_{s \in \mathcal{D}} C_s \); |
| if \( D_s \geq \delta \) then |
| \( S = S \cup C_s \); |

**Analysis:** In Algorithm 1, step 2 partitions the trajectory database into \( n \) equal size parts and each one is distributed into a distinct processor. Steps 3-7 compute the support count of each T-pattern and broadcast the support to other processors. The processors will sum up the support count values, determine whether a T-pattern is a global FT-pattern, and finally output \( G_1 \)-patterns to \( S \) (steps 8-10).

The second phase of parallel processing is the task parallel formulation. This formulation is shown in Algorithm 2. In this phase, we integrate the basic idea of task parallelism into FDMSP, and achieve the trajectory task parallelism. First, we have to partition each \( G_1 \)-subtree into multiple \( G_2 \)-subtrees (step 1). Then, we use the hash partitioned sequential pattern mining algorithm HPSM [14] to assign \( G_2 \)-subtrees to their corresponding processors (steps 2-3). Such processor is called agent that is responsible for calculating the global support of each FT-pattern in a \( G_2 \)-subtree, and determining whether it is globally frequent. The hash function can guarantee that each T-pattern corresponds to one unique agent. In particular, it does not affect the selection of an agent in terms of a T-pattern whether it is frequent in this agent or not [17].

As FDMSP, after the agent receives the set of all \( G_2 \)-subtrees, it first merge such \( G_2 \)-subtrees that have the similar prefix into one single \( G_2 \)-subtree (steps 4-5). The merging approach works as [8, 17]: First, treat each \( G_2 \)-subtree as a union of T-patterns and sum up the support count of the same nodes (T-patterns) in each trajectory set. Second, prune candidate T-patterns by strategies presented in [8].
(step 6). Third, each processor sends the request for computing the support counts of each node in the $G_2$-subtree to other processors (step 8). When the processor receives the requests from its agent, it will compute the support count of the required T-pattern, and sent the results to the corresponding agent. After the agent receives the counting values from all processors (step 9), it finds FT-patterns and outputs $G_2$-patterns to $G$ (steps 10-12). Finally, the agents will generate $G_2$-subtrees by pruning infrequent T-patterns (step 13) and broadcast them to other processors (step 14) simultaneously. We have to repeat the above steps at each processor to achieve the task parallel formulation.

Algorithm 2 Task Parallel Formulation

**Input:** a set of $G_1$-subtree $T$, a minimum support threshold $\delta$, a radius for spatial neighborhoods $\epsilon$.

**Output:** a set of global T-patterns $G$.

1. Partition $T$ into $m$ $G_2$-subtrees;
2. for each $G_2$-subtree $t \in T$ do
3. $A_i \leftarrow \text{HPSPM}(t)$;
4. for each agent $A_i$ do
5. $C_t \leftarrow \text{Merge}()$;
6. Prune candidate T-patterns;
7. for each T-pattern $s \in C_t$ do
8. Send the count request of $s$ to other processors;
9. Receive counting values from other processors;
10. $C_s \leftarrow \text{ComputeDensity}(s, \epsilon)$;
11. if $C_s \geq \delta$ then
12. $G = G \cup C_s$;
13. Generate global $G_2$-subtrees by pruning infrequent T-patterns;
14. Broadcast global $G_2$-subtrees to other processors;

**Analysis:** The complexity of this algorithm is $O(m \ast n)$, where $m$ is the number of processors and $n$ is the number of trajectory patterns. The benefit of this parallel formulation approach lies in that it utilizes parallelism characteristics to achieve the communication and the mining simultaneously, which help improve the efficiency of computations.

## 5 Experiments and Discussions

### 5.1 Experimental setup

We report the experimental studies by comparing PartSpan with two representative parallel sequence mining algorithms, i.e., distributed GSP algorithm (DGSP) that simply extends the GSP algorithm to adapt to the distributed system, and parallel SPADE algorithm (pSPADE) [16]. The experiments are performed on a computer group consisting of 8 PC with Pentium IV 2.4 GHz CPU, 512 Mb of RAM. All algorithms were run on the following real-world and synthetic datasets, respectively.

1. The real Truck dataset consists of 276 trajectories of 50 trucks that deliver concrete to several construction places around Athens metropolitan area in Greece for 33 distinct days, for a total of 112,203 points.

2. The synthetic data are generated by Brinkhoff’s network-based generator [3]. It contains 100,000 trajectories of one day movement over the road-network of the city of Oldenburg. The cardinality of the data is about 225Mb.

### 5.2 Parallel time comparison

The experimentations consist of evaluating the execution time under a varying number of processors. We compare PartSpan with other parallel algorithms by changing the number of processors from 4 to 8. Figs.1-2 show the execution time across distinct algorithms in the real-world and the synthetic datasets at a minimum support of 0.1%.

*Download at http://www.rtreeportal.org/*
By Fig. 1 and Fig. 2, the execution time of DGSP keeps almost unchanged as the number of processors grows. Since DGSP assigns the equal size data to multiple processors, the execution time are not affected by the number of processors. Whereas, the runtime of pSPADE and PartSpan decreases as the number of processors increases. This is due to the specific parallel formulations. In particular, our proposed PartSpan is the winner in any case of distinct processors, and improve the time performance with respect to DGSP by a factor up to 6.8 in the real-world dataset, and a factor of 2 in the synthetic dataset. PartSpan achieves time saving improvement with regard to pSPADE of 1.1-2 times in the real-world data, and of 0.4-0.5 times in the synthetic data. This is because PartSpan employs agents to find global FT-patterns that helps reduce the communication cost. In addition, it uses the prefix-projection approach to reduce candidate T-patterns before sending $G_2$-subtrees to its agent.

6 Conclusions and Future Directions

In this paper, we propose a novel, parallel, and efficient trajectory pattern mining algorithm with time constraints, called PartSpan. Its general idea is to partition the search space by the prefix-projection approach, and to introduce the parallel strategy to divide parallel computations into the data formulation and the task formulation. To further improve the mining efficiency, specific candidate strategies are applied. The performance study describes in detail the merits of PartSpan with regard to varying number of processors in the real-world as well as the synthetic datasets.

PartSpan is a promising and new methodology at efficiently mining trajectory patterns in very large databases, it can also be directly applied to mining other sequential patterns with time annotations. Our future work includes: (i) extending our parallel algorithm to mining web logs, (ii) optimizing important parameters, such as the support for parallel computations, and (iii) developing a prediction method of uncertainty trajectories in moving databases, since the real-world time annotated data is difficult to obtain.

References