Hindawi Publishing Corporation
Journal of Sensors
Volume 2015, Article ID 461659, 13 pages
http://dx.doi.org/10.1155/2015/461659

Research Article

An Asynchronous Periodic Sequential Pattern Mining Algorithm with Multiple Minimum Item Supports for Ad Hoc Networking

Xiangzhan Yu,[1] Zhaoxin Zhang,[1] Haining Yu,[1] Feng Jiang,[1] and Wen Ji[2]

[1]*School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China*
[2]*Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China*

Correspondence should be addressed to Zhaoxin Zhang; heart@hit.edu.cn

Received 9 January 2015; Accepted 28 February 2015

Academic Editor: Daniel Bo-Wei Chen

The original sequential pattern mining model only considers occurrence frequencies of sequential patterns, disregarding their occurrence periodicity. We propose an asynchronous periodic sequential pattern mining model to discover the sequential patterns that not only occur frequently but also appear periodically. For this mining model, we propose a pattern-growth mining algorithm to mine asynchronous periodic sequential patterns with multiple minimum item supports. This algorithm employs a divide-and-conquer strategy to mine asynchronous periodic sequential patterns in a depth-first manner recursively. We describe the process of algorithm realization and demonstrate the efficiency and stability of the algorithm through experimental results.

1. Introduction

Ad hoc networking has become a hot issue recently as large-scale nodes exhibit in the network. Heterogeneous devices and heterogeneous environments deepen the difficulty of ad hoc networking. The sequential pattern mining model is used to discover all sequential patterns that occur frequently in a sequence database; the target database is shown in Table 1. The spatiotemporal sequence mining model is used to discover all frequent spatiotemporal sequential patterns in a spatiotemporal sequence database; the target database is depicted in Table 2. The periodic pattern mining model is used to discover all periodic patterns in a temporal sequence database; the target database is shown in Table 3.

According to the three mining models, the time property of a sequence can be divided into two levels. (1) Fine-grained time property is as follows: the fine-grained time property is used to annotate each item set of each sequence, such as the subscript t_i of every item set in Table 4. (2) Coarse-grained time property is as follows: the coarse-grained time property is used to annotate each sequence, such as *tid* in Table 4. The fine-grained time property is a further division of the coarse-grained time property. The sequential pattern mining model only uses the fine-grained time property to sort items and ignores the coarse-grained time property of a sequence. The fine-grained time property is utilized in the spatiotemporal sequence mining model. This model uses the temporal annotation as a criterion to judge whether two sequences are equal, and deviations of the annotation can be tolerated. Nevertheless, both models only concentrate on the frequency of sequences but ignore the characteristics of the distribution of sequences in a database. The periodic pattern mining model takes the fine-grained time property as a criterion to discover periodic patterns, and it uses periodic patterns within the fine-grained time property to calculate the patterns within the coarse-grained time property. However, the restrictions of the temporal annotations of item sets are too strict (the fine-grained time property should be divided into unit time intervals); besides, frequent deviations of annotations of item sets would not be permitted while mining. In addition, a myriad of sparse periodic patterns with little meaning will be generated by the periodic pattern mining model when the time series is sparse.

The original sequential pattern mining model only considers the occurrence frequencies of sequential patterns and disregards their occurrence periodicity. Therefore, we propose an asynchronous periodic sequential pattern mining model, and, for this model, we propose a pattern-growth

Table 1: Symbolic sequence database.

ID	Sequence
0	$\langle a\ (abc)\ (ac)\ d\ (cf)\rangle$
1	$\langle (ad)\ c\ (bc)\ (ae)\rangle$
2	$\langle (ef)\ (ab)\ (df)\ c\ b\rangle$
3	$\langle e\ g\ (af)\ c\ b\ c\rangle$

Table 2: Spatiotemporal sequence database.

ID	Sequence
0	$\langle a_{t0}\ (abc)_{t1}\ (ac)_{t2}\ d_{t3}\ (cf)_{t4}\rangle$
1	$\langle (ad)_{t0}\ c_{t1}\ (bc)_{t2}\ (ae)_{t3}\rangle$
2	$\langle (ef)_{t0}\ (ab)_{t1}\ (df)_{t2}\ c_{t3}\ b_{t4}\rangle$
3	$\langle e_{t0}\ g_{t1}\ (af)_{t2}\ c_{t3}\ b_{t4}\ c_{t5}\rangle$

Table 3: Temporal sequence database.

Temporal sequence
$a_{t00}\ (abc)_{t01}\ (ac)_{t02}\ d_{t03}\ (cf)_{t04}\ (ad)_{t10}\ c_{t11}\ (bc)_{t12}\ (ae)_{t13}$
$(ef)_{t20}\ (ab)_{t21}\ (df)_{t22}\ c_{t23}\ b_{t24}\ e_{t30}\ g_{t31}\ (af)_{t32}\ c_{t33}\ b_{t34}\ c_{t35}$

Table 4: Fine and coarse grained temporally annotated sequence database.

tid	Sequences with temporal annotations
tid_0	$\langle a_{t0}\ (abc)_{t1}\ (ac)_{t2}\ d_{t3}\ (cf)_{t4}\rangle$
tid_1	$\langle (ad)_{t0}\ c_{t1}\ (bc)_{t2}\ (ae)_{t3}\rangle$
tid_2	$\langle (ef)_{t0}\ (ab)_{t1}\ (df)_{t2}\ c_{t3}\ b_{t4}\rangle$
tid_3	$\langle e_{t0}\ g_{t1}\ (af)_{t2}\ c_{t3}\ b_{t4}\ c_{t5}\rangle$

mining algorithm to mine asynchronous periodic sequential patterns with multiple minimum supports. The algorithm is a recursive algorithm that uses a divide-and-conquer strategy to mine the patterns, and the search is depth-first.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of recent related research on sequential pattern mining and periodic sequential pattern mining. In Section 3, definitions of related concepts are introduced. In Section 4, the asynchronous and synchronous periodic sequential pattern mining models are presented. Section 5 proposes a pattern-growth mining algorithm to mine asynchronous periodic sequential patterns with multiple minimum item supports. In Section 6, the experimental results show the efficiency and stability of the algorithm. We present the conclusions of our study in Section 7.

2. Related Work

Traditional sequential pattern mining algorithms, such as FreeSpan, PrefixSpan [1], and SPADE [2], discover frequent subsequences as patterns in a sequence database. These traditional algorithms are described in detail in [3]. A spatiotemporal sequence mining algorithm is a special type of sequence mining algorithm. Deviations of temporal annotation and spatial location of a sequence should be considered in this method. In [4], a spatiotemporal sequence mining algorithm based on PrefixSpan is proposed. In this algorithm, temporal annotations not only are used to sort the location or status but also are involved in mining the spatiotemporal sequence directly. However, the complexity of the algorithm would increase dramatically as the spatial dimension aggrandizes.

Traditional periodic pattern mining algorithms discover periodic patterns in time-related databases, and these algorithms can be divided into several categories as follows by different characteristics:

(1) the full periodic pattern mining method and partial periodic pattern mining method: full periodic patterns [5] are those in which all items of the time series take part in the periodic behaviour patterns, whereas there are almost no such patterns due to fairly strict constraints; in partial periodic patterns [6], only a portion of the items of the time series reflects the periodicity; compared with full periodic patterns, partial periodic patterns are more loose and realistic;

(2) the synchronous periodic pattern mining method and asynchronous periodic pattern mining method: synchronous periodic patterns [6] are those in which if one pattern appears at time t_i, it would definitely appear at $t_i + period * n$ ($n > 0$), where period represents the length of the period, and the patterns, which do not happen at such fixed times, would be taken as irrelevant patterns; asynchronous periodic patterns [7] are those in which if one pattern appears at time t_i, instead of just taking place at $t_i + period * n$ ($n > 0$), the pattern would appear at any time of the time series; synchronous periodic patterns are a special case of asynchronous periodic patterns, and the latter is more realistic.

The full periodic pattern mining method has been widely studied in the field of signal analysis. Fast Fourier transforms and Wavelet analysis are often used to find the full periodic patterns in time series data. In [8], a partial periodic pattern mining algorithm based on the downward closure property is proposed. To improve efficiency, this algorithm builds a max-subpattern tree to separate partial periodic patterns. The authors in [9] proposed a convolution-based algorithm employing the improved fast Fourier transform to mine the partial periodic patterns and discover all possible synchronous periodic patterns. Time series are divided into intensive intervals in [10], and then the synchronous periodic patterns are mined. [7, 11] propose an asynchronous periodic sequential pattern mining algorithm called LSI to find the longest periodic subsequence, in which some sequences whose lengths are less than the threshold value can exist. However, the algorithm is not suitable for the condition of multievents, and only the longest subsequence of the asynchronous periodic patterns can be found, with other subsequences being ignored. Aiming to address these shortcomings, an improved algorithm, SMCA, based on a hash table and enumeration, is proposed by [12]. This algorithm not only implements all functions of LSI but also corrects the defects and improves the efficiency. Reference [13] proposes

an algorithm for mining periodic patterns that utilizes the method of [3] to preprocess trajectory sequences and uses the max-subpattern tree proposed by [8] to discover periodic patterns. Reference [14] proposes a method to mine periodic-frequent item sets with approximate periodicity using an interval transaction-ids list tree, and it is extended by [15] to fulfill the requirement of mining periodic-frequent patterns in transactional databases. Reference [16] proposes a HACE theorem that characterizes the features of the Big Data revolution and proposes a Big Data processing model, from the data mining perspective. This data-driven model involves demand-driven aggregation of information sources, mining and analysis, user interest modeling, and security and privacy considerations. Reference [17] proposes a trajectory prediction approach for mobile objects by combining semantic features through pattern mining in the geographic trajectory data of user. Reference [18] focuses on dynamic networks to study the related pattern mining method and its applications in biological and social networks. Reference [19] proposes a pattern mining algorithm which is called Multiconstraint Closure Conditional Tree. The algorithm sets different limiting conditions to solve the combinatorial explosion problem and the rare item problem. And, to prevent noise and other uncertainties, this paper introduces similarity-based pattern matching method, making the pattern mining method more robust.

3. Basic Concept and Definition

Before talking about the asynchronous periodic sequential pattern mining algorithm, some basic concepts will be defined.

At first, we assume that $I = \{i_1, i_2, \ldots i_n\}$ is the set of all items, and each item is the nonempty subset of I. A sequence is a sequential list of item sets, and it can be expressed as $s = \langle s_1, s_2, \ldots, s_l\rangle$, where s_j is an item set and also en element of s, which can be expressed as $s_j = (x_1, x_2, \ldots, x_m)$, where x_k is an item. To simplify the expression, the parentheses can be omitted if an item only consists of one element; for example, "(x)" can be expressed as "x." One item can appear merely once or less in one element of a sequence, but more than one item can exist in different elements of a sequence. In general, the elements are sorted lexically, and the number of elements is the length of the sequence. A sequence of length l is called an l-sequence.

The target database of asynchronous periodic sequential pattern mining is defined as follows.

Definition 1 (temporal symbolic sequence database). The temporal symbolic sequence database is a set of tuples; that is, $TDB = \{t_0, t_1, \ldots t_{|TDB|-1}\}$, where $|TDB|$ is the number of tuples and t_i represents the tuple (tid, s), where tid is the time when symbolic sequence s takes place. In general, tuples of TDB are sorted in ascending order, and each tid is evenly spaced. Figure 1 shows a temporal symbolic sequence that consists of 23 tuples.

Definition 2 (subsequence and supersequence). Sequence $\alpha = \langle a_1 a_2 \cdots a_n\rangle$ is a subsequence of sequence $\beta = \langle b_1 b_2 \cdots b_m\rangle$, and β is the supersequence of α, which can be expressed as $\alpha \sqsubseteq \beta$ if there exists a set of integers $1 \le j_1 < j_2 < \cdots < j_n \le m$, where $a_1 \subseteq b_{j_1}$ and $a_2 \subseteq b_{j_2}, \ldots, a_n \subseteq b_{j_n}$.

Definition 3 (containment and appearance). $t' = (tid', s')$ is one of the tuples of TDB, and α is a sequence. We can say that t' contains α or α appears at t' only if α is a subsequence of s'.

For instance, as is shown in Figure 1, $\langle(ab)adf\rangle$ appears at the tuple $(0, \langle a(abc)(ac)d(cf)\rangle)$.

Definition 4 (prefix). Sequence $\beta = \langle e'_1 e'_2 \cdots e'_m\rangle$ is the prefix of sequence $\alpha = \langle e_1 e_2 \cdots e_n\rangle$ $(m \le n)$ only if these three requirements are all met: (i) $e'_i = e_i$, where $(i \le m - 1)$. (ii) $e'_m \subseteq e_m$. (iii) All items of $(e_m - e'_m)$ are after e'_m by lexicographic order.

Definition 5 (postfix). Sequence $\beta = \langle e_1 e_2 \cdots e_{m-1} e'_m\rangle$ is the prefix of sequence $\alpha = \langle e_1 e_2 \cdots e_n\rangle$ $(m \le n)$, and sequence $\gamma = \langle e''_m e_{m+1} \cdots e_n\rangle$ is called the postfix of α with regard to prefix β, denoted as $\gamma = \alpha/\beta$ or $\alpha = \beta \cdot \gamma$, where $e''_m = (e_m - e'_m)$.

Definition 6 (periodic projected database). Construct a temporal symbolic sequence database TDB, and specify a preset parameter of maximum period max_per, $max\_per \ll (tid_f - tid_l)$, where tid_f is the first time stamp of TDB and tid_l is the last time stamp. The periodic projected data of TDB that starts at time l with a period p is composed by the tuple (tid_{l+i*p}, s_{l+i*p}) $(0 \le i \le m)$, where $l < p$, $1 \le p \le max\_per$, and $m = \lceil (tid_l - l)/p \rceil$, and these data can be expressed as $prj_{p,l}(TDB)$.

For instance, as is shown in Figure 1, the tuples of $prj_{3,1}(TDB)$ are as follows: 1: $\langle(bd)c(bc)(ad)\rangle$, 4: $\langle bc(bc)(ad)\rangle$, 7: $\langle c(bc)ab(abd)\rangle$, 10: $\langle ac(abc)(ad)\rangle$, 13: $\langle(ac)(abe)cb\rangle$, 16: $\langle(cef)a(ab)ddb\rangle$, and 19: $\langle b(ac)(abd)\rangle$.

Definition 7 (prefix projected databases). If sequence α is an asynchronous periodic sequential pattern in TDB, then the prefix projected database is the subdatabase that is composed of all of the prefixes of α with regard to the postfix of α, denoted as $S|_\alpha$.

Definition 8 (contained segments). Sequence α with a period of p in TBD contains a segment L, which is a time list including consecutive tuples of α in $prj_{p,l}(TDB)$, where the timestamp l is the remainder of the division of the starting position by p. The contained segment L consists of a quaternion $(\alpha, p, rep, begin)$, where α is the target sequence, p is the period, rep is the number of times in which α appears, and $begin$ is the starting position where the repeated appearance of α starts. L can be called the maximum contained segment if α is not contained in the two directions of L.

As is shown in Figure 1, the maximum contained segment L_2 is expressed as $(\langle a(ab)\rangle, 3, 5, 7)$, which means that $\langle a(ab)\rangle$ starts at tid 7 and appears repeatedly a total of 5 times with

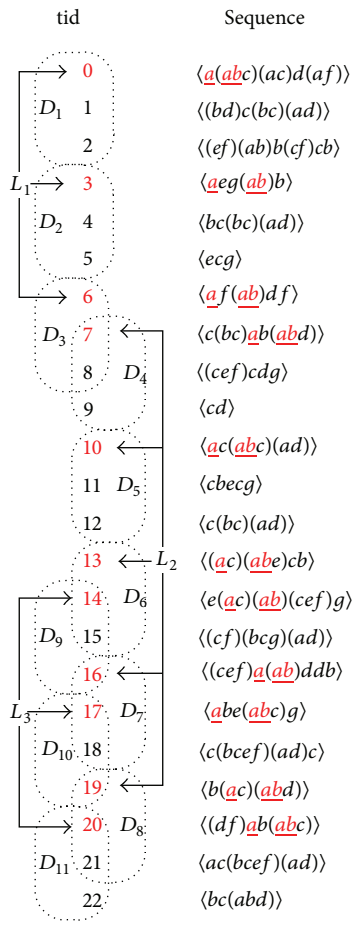

Figure 1: Temporal symbolic sequence database.

a period of 3 and the *tid*s at which $\langle a(ab)\rangle$ appears are 7, 10, 13, 16, and 19. Similarly, L_1 is expressed as ($\langle a(ab)\rangle$, 3, 3, 0), L_3 is expressed as ($\langle a(ab)\rangle$, 3, 3, 14), and both of them are the maximum contained segments with a period of 3.

Definition 9 (valid contain segments). A maximum contained segment $L = (\alpha, p, rep, begin)$ is valid only if the repeat count of L is not less than *min_rep*; that is, $L.rep \geq min\_rep$.

As is shown in Figure 1, we assume that *min_rep* = 3; then, the maximum contained segments L_1, L_2, and L_3 are the valid contained segments of $\langle a(ab)\rangle$ with a period of 3 because the repeat counts of L_1, L_2, and L_3 are 3, 5, and 3, respectively, which are not less than 3.

Definition 10 (distance between contained segments). Given two contained segments, L and L' ($L.begin < L'.begin$), the distance, *dis*, between them is the difference between the starting time of L' and the ending time of L; that is, $dis = L'.begin - [L.begin + p * (L.rep - 1)]$. If $dis < 0$, we can say that L and L' intersect; otherwise, they do not.

As is shown in Figure 1, the distance between L and L' is 1 (7 – 6), the distance between L_1 and L_3 is 8 (14 – 6), and the distance between L_2 and L_3 is –5 (14 – 19). Therefore, we can safely draw the conclusions that (L_1, L_2) and (L_1, L_3) are not intersecting, while (L_2, L_3) is intersecting.

Definition 11 (sequences of contained segments). The sequence of contained segments is the sequence that meets the following requirements: (i) contained segments come from the same database and are of the same sequence with the same period; (ii) contained segments must be valid; (iii) contained segments are sorted by increasing starting time; (iv) any two contained segments are not intersecting. The sequence of contained segments of sequence α with a period of p can be expressed as follows:

$$Q_p(\alpha) = \left(L_1 \xrightarrow{dis_1} L_2 \xrightarrow{dis_2}, \ldots, \xrightarrow{dis_{n-1}} L_n\right), \quad (1)$$

where L_i is a valid segment, $L_i.begin < L_{i+1}.begin$ ($1 \leq i \leq n-1$) and the distance dis_i between two consecutive segments can be expressed by the following equation: $dis_i = L_{i+1}.begin - [L_i.begin + p * (L_i.rep - 1)]$.

Definition 12 (valid sequences of contained segments). The sequence of contained segments $Q_p(\alpha) = (L_1 \xrightarrow{dis_1} L_2 \xrightarrow{dis_2}, \ldots, \xrightarrow{dis_{n-1}} L_n)$ is valid when $dis_i \leq max\_dis * p$ and $\sum_{i=1}^{n} L_i.rep \geq min\_sup$, where *max_dis* indicates the preset maximum distance coefficients and *min_sup* is the preset minimum support of the asynchronous period.

Figure 1 shows that, in the symbolic database, if *min_rep* = 3, *max_dis* = 3, and *min_sup* = 6, then we can say that ($L_1 \rightarrow L_2$) and ($L_1 \rightarrow L_3$) are valid sequences of the contained segments of $\langle a(ab)\rangle$ with a period of 3.

4. Periodic Sequential Pattern Mining Model

4.1. Synchronous Periodic Sequential Pattern Mining Model. In a temporal symbolic sequence database *TDB*, the synchronous periodic support of sequence α with a period p starting at *tid* l is the number of tuples that α contains. This is shown as follows:

$$\text{support}_{TDB}^{p,l}(\alpha) = \left|\left\{(tid, s) \mid \left((tid, s) \in prj_{p,l}(TDB)\right) \wedge (\alpha \sqsubseteq s)\right\}\right|. \quad (2)$$

Given a minimum support *min_sup* ($0 < min\_sup \leq 1$) and a maximum period *max_per* ($0 < max\_per \ll |TDB|$), sequence α in *TBD*, which starts at time l, is frequent with the period p if $\text{support}_{TDB}^{p,l}(\alpha) \geq min\_sup$ and $0 < p \leq max\_per$. A synchronous periodic-frequent sequence is called a synchronous periodic sequential pattern. The goal of a synchronous periodic sequential pattern mining algorithm is to discover all synchronous periodic sequential patterns whose periods are from 1 to *max_per*.

4.2. Asynchronous Periodic Sequential Pattern Mining Model. In many applications, the periodicity of a sequence is usually not perfect and precise, and there may exist noise between segments of a sequence. However, only the noise of a

"sequential deficiency," not the noise of a "sequential offset," can be recognized by this model; as a result, many sequential patterns with high value cannot be discovered. To some degree, the interruption of noise can be tolerated by the periodic sequential mining algorithm. In addition, "system behaviour" occurs repeatedly, and, then, it disappears or changes. The sequential patterns with such uncertainty only appear periodically in a portion of *TDB*.

Based on the analysis above, this paper proposes an asynchronous periodic sequential pattern mining model that can discover the patterns that appear periodically in various time periods of the *TDB*; also, certain noise can be tolerated in this model. The main idea of the model is as follows. At first, to judge if a sequence is a potential asynchronous periodic sequential pattern, the sequence must appear repeatedly, which means that this sequence has significant periodic trends. Then, the time interval within which the sequence appears periodically would be examined to determine whether the time interval is "random noise" or "the change of system behaviour." Finally, on the premise that the noise is tolerated, the time periods at which the sequence appears periodically would be linked to obtain the maximum periodic time range.

Concretely, an asynchronous periodic sequential pattern and its mining model are defined as follows.

Definition 13 (asynchronous periodic sequential pattern). Sequence α is one asynchronous periodic sequential pattern in the temporal symbolic database *TDB*. If there exists more than one valid sequence of contained segments relating to α, then the period of such sequences is that of α. It is not difficult to find that one asynchronous periodic sequential pattern may have several periods, and each period may involve several valid sequences of contained segments.

Definition 14 (asynchronous periodic sequential pattern mining model). Given the minimum support, the maximum distance coefficients, the minimum repeat count, and the maximum period, the purpose of asynchronous periodic sequence mining is to discover all asynchronous periodic sequential patterns and their valid sequences of contained segments in *TDB*.

5. AP-PrefixspanM Mining Algorithm

To use the one and only minimum support, we must assume that all the items in the database have the same properties and a similar frequency of occurrence. However, this assumption conflicts with the actual application situation, which leads to the result that the sequential patterns with few but important items are omitted. A perfect asynchronous periodic sequential pattern mining algorithm should support multiple minimum item supports, which means users can identify each item with minimum support, and different requirements of minimum supports can be met by different sequences with different items. With multiple minimum item supports, not only can we prevent the generation of a myriad of meaningless asynchronous periodic sequential patterns, but we can also discover the sequential patterns with few items.

In this paper, we propose a pattern-growth mining algorithm to mine asynchronous periodic sequential patterns with multiple minimum item supports; this algorithm is called AP-PrefixspanM (asynchronous periodic prefix projected sequential patterns with multiple minimum item supports).

5.1. Relationship between Algorithm Parameters. Let $MinIS(i)$ represent the minimum support of item i. The minimum support of the sequential pattern P is the lowest minimum support of all items of P; for example, if the item set of P is $i_1, i_2, \ldots, i_k$, then the equation of minimum support of P is

$$\begin{aligned} &min\_sup\,(P) \\ &\quad = min\left(MinIS\left(i_1\right), MinIS\left(i_2\right), \ldots, MinIS\left(i_k\right)\right). \end{aligned} \tag{3}$$

Let $MinREP(i)$ be the minimum repeat count of item i; the minimum repeat count of P is the lowest repeat count of all items; that is

$$\begin{aligned} &min\_rep\,(P) \\ &\quad = min\left(MinREP\left(i_1\right), MinREP\left(i_2\right), \ldots, MinREP\left(i_k\right)\right). \end{aligned} \tag{4}$$

Additionally, let $MaxDis(i)$ be the maximum interference distance of item i, which represents a reasonable bound of disturbance between two valid sequences of contained segments; the maximum interference distance of P is the largest maximum interference distance of all items; that is

$$\begin{aligned} &max\_dis\,(P) \\ &\quad = max\left(MaxDis\left(i_1\right), MaxDis\left(i_2\right), \ldots, MaxDis\left(i_k\right)\right). \end{aligned} \tag{5}$$

To reduce the workload of user settings, we assume that there is a function relationship between the maximum interference distance, the minimum repeat count, and the minimum support. Specifically, the relationship between the minimum repeat count of item i, $MinREP(i)$, and the minimum support of item i, $MinIS(i)$, is linear increasing, and it can be expressed as follows:

$$MinREP\,(i) = \lceil \lambda * MinIS\,(i) \rceil \quad (0 < \lambda \leq 1). \tag{6}$$

The relationship between the maximum interference distance of item i, $MaxDis(i)$, and the minimum support of item i, $MinIS(i)$, is exponential decreasing, and it can be expressed as follows:

$$MaxDis\,(i) = \left\lceil \frac{\eta * \max\,(MinIS)}{MinIS\,(i)} \right\rceil, \tag{7}$$

where η is the preset constant of interference distance and max ($MinIS$) is the largest minimum support of $MinIS$.

If $MinIS(i_1) \leq MinIS(i_2) \leq \cdots \leq MinIS(i_k)$, then $MinREP(i_1) \leq MinREP(i_2) \leq \cdots \leq MinREP(i_k)$, $MaxDis(i_1) \geq MaxDis(i_2) \geq \cdots \geq MaxDis(i_k)$.

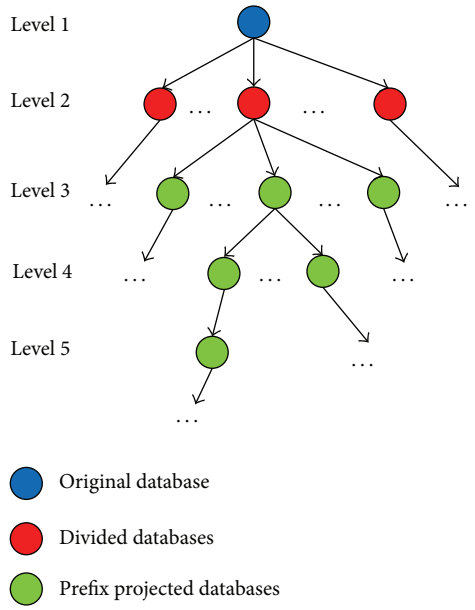

Figure 2: The framework of database division in the AP-PrefixspanM algorithm.

5.2. The Main Idea and Steps of the Algorithm. For the purpose of reducing the search space of asynchronous periodic symbolic sequential pattern mining, the downward closure property is used in the asynchronous periodic sequential pattern mining model with single minimum support.

Property (Downward Closure Property). If the sequence α is an asynchronous periodic sequential pattern and its set of valid sequences of contained segments is *QSet*, then we can say that all the nonempty subsequences of α are also asynchronous periodic sequential patterns, and the set of valid sequences of contained segments of these subsequences is the superset of *QSet*.

However, because the minimum item support of an asynchronous periodic sequential pattern may be less than its subsequence's with multiple minimum item supports, the downward closure property cannot be used directly to prune the search space.

The main idea of the *PrefixspanM* algorithm is based on the divide-and-conquer strategy. The problem of asynchronous periodic sequential pattern mining is divided progressively into a series of subproblems that are not intersecting. The framework of database division in the *PrefixspanM* algorithm is as shown in Figure 2. Two dividing methods can be adopted: (1) prune the search space with the downward closure property: at level 1, the database is divided into a series of subdatabases by the minimum support vectors and (2) generate asynchronous periodic sequential patterns with pattern-growth: below the second level, the prefix projected database is generated recursively by the prefix.

Firstly, the *AP-PrefixspanM* algorithm divides the database based on minimum support vectors. Then, a series of subdatabases are generated and mined by the single minimum support. Each division is based on a frequent item called a key item. The single minimum support utilized is called the minimum support of the key item. The downward closure property will not be destroyed while mining, and the specific splitting steps of data are described as follows.

Step 1. Scan the database TDB and obtain item x_i, whose real support is at least $MinIS(x_i)$. Such items x_i are called frequent items, and they are placed in ascending order according to their *MinIS* to obtain $list = x_1, x_2, \ldots, x_n$, $(x_i \leq x_{i+1})$.

Step 2. The complete set of sequential patterns of asynchronous period in *TDB* can be divided into the following n mutually disjoint subsets, among which some subsets may be empty:

(1) sequential patterns that contain sequence x_1,

(2) sequential patterns that contain sequence x_2 but do not contain x_1,

$\vdots$

(i) sequential patterns that contain sequence x_i but do not contain $\{x_1, x_2, \ldots, x_{i-1}\}$,

$\vdots$

(n) sequential patterns that only contain x_n.

Divide *TDB* by these n subsets, and then mine the n subdatabases to obtain the subsets of these n asynchronous periodic sequential patterns:

(1) TDB_1 (key item x_1): delete the infrequent items of tuples and the tuples that do not contain x_1;

(2) TDB_2 (key item x_2): delete the infrequent items of tuples, x_1, and the tuples that do not contain x_2;

$\vdots$

(i) TDB_i (key item x_i): delete the infrequent items of tuples, $\{x_1, x_2, \ldots, x_{i-1}\}$, and the tuples that do not contain x_i;

$\vdots$

(n) TDB_n (key item x_n): delete all the infrequent items of tuples except for x_n and the tuples that do not contain x_n.

Mine the asynchronous periodic sequential patterns in the TDB_i with the minimum support of $MinIS(x_i)$. This problem can be divided into a series of mutually disjoint subproblems.

(1) Let $\{\langle y_1\rangle, \langle y_2\rangle, \ldots, \langle y_m\rangle\}$ be the complete set of length-l asynchronous periodic sequential patterns in TDB_i, and this complete set can be divided into m mutually disjoint subsets. The subset j $(1 \leq j \leq m)$ is the set of asynchronous periodic sequential patterns whose prefix is $\langle y_j\rangle$.

(2) Let α be the asynchronous periodic sequential length-l pattern. $\{\beta_1, \beta_2, \ldots, \beta_t\}$ is the set of asynchronous

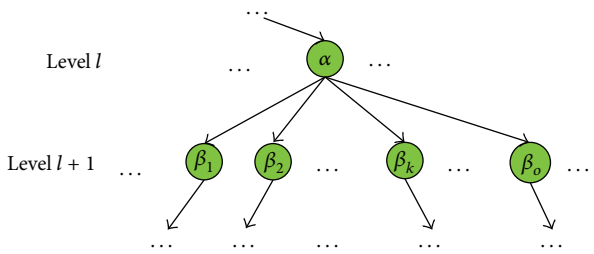

Figure 3: Division of a subset of asynchronous periodic sequential patterns.

> periodic sequential patterns whose prefix is α and length is $l + 1$. Apart from α, the complete set of asynchronous periodic sequential patterns whose prefix is α can be divided into k mutually disjoint subsets. The subset k ($1 \leq k \leq t$) is the subset of asynchronous periodic sequential patterns whose prefix is β_k, as is shown in Figure 3.

Based on the descriptions above, the problem of asynchronous periodic sequential pattern mining can be divided recursively, which suggests that each subset can be further divided. Thus, a divide-and-conquer framework is composed. To mine the subsets of asynchronous periodic sequential patterns, we can construct corresponding prefix projected databases and mine each database recursively. The main step is shown as follows:

Step 1 (generate parameters). Regarding *MinIs* as the minimum support vector, generate the minimum repeat count vector, *MinREP*, and the maximum interface distance vector, *MaxDis*.

Step 2 (find the length-1 sequential patterns). Scan the database once and generate all length-1 asynchronous periodic sequential patterns. The length of all patterns will be increased by 1 when the prefix is extended by those length-1 sequential patterns.

Step 3 (divide the search space). Divide the complete set of asynchronous periodic sequential patterns into the subsets whose prefixes are the patterns whose lengths have been increased by 1.

Step 4. Construct the prefix projected databases, and, then, discover the subsets of each asynchronous periodic sequential pattern recursively. Specifically, repeat the steps above until no length-1 asynchronous periodic sequential patterns can be generated.

In the mining process described above, only the asynchronous periodic sequential patterns will be added to the mining result. Other patterns will be deleted after mining.

5.3. Implementation of the Algorithm. The pseudocode of *APPrefixspanM* is described in Algorithm 1. Firstly, regarding *MinIS* as the minimum support vector, generate the minimum repeat count *MinREP* and the maximum interference distance *MaxDIS* (as is described in steps (1) and (2) of Algorithm 1).

Then, scan the *TDB* once and find all the frequent items x_i for which the real support is more than $MinIS(x_i)$. Such items are the potential asynchronous periodic sequential patterns whose lengths are 1 (as is described in step (4) of Algorithm 1). Obtain the list SPF_1 by sorting the frequent items in ascending order based on the minimum support vector *MinIS* (as is described in step (5) of Algorithm 1). Finally, obtain the $|SPF_1|$ divided subdatabases, and, for each sub database, call the function *MAPPrefixSpan* to mine the asynchronous periodic sequential patterns (as is described in steps (6)–(10) of Algorithm 1).

The function *MAPPrefixSpan* discovers all synchronous periodic sequential patterns by the method of depth-first search (as is described in Algorithm 2). First, we define a hash map *hash_item* ⟨*item_id*, (*count*, *tids*)⟩ to record each frequent item's frequency and its time slot (as is described in step (1) of Algorithm 2), where *item_id* is the key of the hash map, two-tuples (*count, tids*) is the value, *count* is the frequency of *item_id*, and *tids* is the time slot queue of *item_id*. Each item x has two possible ways to be extended to the prefix and obtain a new sequential pattern: (1) join x to the last item set of the prefix and *item_id* of x is expressed as "*_x*," and (2) join x independently to the prefix, and *item_id* of x is expressed as "*x*." Scan the projected database *TDB*, and record the frequency and the time slot queues of the two extended ways by the hash map (as is described in steps (2)–(8) of Algorithm 2). Delete the items for which the frequency is less than *min_sup* in the hash map (as is described in steps (9)–(11) of Algorithm 2). At this time, the frequent items of *hash_item* may be extended to the current prefix, and a new asynchronous periodic sequential pattern would be generated. For each item, call the function *ASPDetector* to calculate the possibility of frequent items being extended to the prefix to generate new patterns. If there would be new patterns, then all the valid contained segment queue set of the new pattern would be generated (as is described in step (13) of Algorithm 2). If frequent items can be extended to the prefix, then generate the new prefix after extension (as is described in step (15) of Algorithm 2); meanwhile, call the function *Projectdatabase* to generate the prefix projected database $DB|_c$ of the current item. If *c_item* is not included in the prefix, delete the tuple in which the key item *c_item* is not included in $DB|_c$ (as is described in steps (16)–(18) of Algorithm 2). If the number of tuples in the filtered $DB|_c$ is not less than *min_sup*, then call the function *MAPPrefixSpan* recursively to discover the items that can be extended to the prefix in the smaller prefix projected database, and discover the asynchronous periodic sequential patterns that are growing progressively (as is described in steps (19)-(20) of Algorithm 2). In the process of the recursion above, all the asynchronous periodic sequential patterns and their valid contained segment queues can be discovered.

The function *ASPDetector* is used to judge whether the current item can be extended to the prefix, obtain a new growing asynchronous periodic sequential pattern (a new

Input: target database *TDB*, minimum support vector *MinIS* and the maximum period *max_per*.
Output: all the asynchronous periodic sequential patterns and valid contained segments queue.
(1) $MinREP \leftarrow$ generateREP(*MinIS*);
(2) $MaxDis \leftarrow$ generateDIS(*MinIS*);
(3) $L \leftarrow init\_pass(M, TDB)$;
(4) $PF_1 \leftarrow \{\langle x\rangle | x \in L, x.count \geq MinIS(x)\}$;
(5) $SPF_1 \leftarrow sort\ (PF_1, \text{ASCEND})$;
(6) **FOR** each $\langle x_i\rangle$ in SPF_1 **DO**
(7) $//beforePF_1^{\langle x_i\rangle}$ is the 1-pattern before $\langle x_i\rangle$ of SPF_1.
(8) $SUB\_TDB_{\langle x_i\rangle} \leftarrow \{t | t \in TDB \wedge \langle x_i\rangle \in t\}$ delete $beforePF_1^{\langle x_i\rangle}$;
(9) $MAPPrefixSpan(null, SUB\_TDB_{\langle x_i\rangle}, \langle x_i\rangle, MinIS(x_i), MaxREP(x_i), MaxDis(x_i))$;
(10) **ENDFOR**

Algorithm 1: *AP-PrefixspanM* (*TDB*, *MinIS*, and *max_per*).

Input: prefix *prefix*, temporal symbolic sequence databases *TDB*, key item *c_item*, minimum support *min_sup*, minimum repeat count *min_rep*, maximum period *max_per*, maximum interference distance *max_dis*.
(1) *hash_item* ⟨*item_id*, (*count, tids*)⟩;
(2) **FOR** each tuple *t* in *DB* **DO**
(3) **IF** $\langle x\rangle \in t.s, x \in L$ **THEN**
(4) $hash\_item[x].count{+}{+}$; $hash\_item[x].tid \leftarrow \cup t.tid$;
(5) **IF** $\langle\{perfix.litemset, x\}\rangle$ *or* $\langle \_x\rangle \in t.s, x \in L$ **THEN**
(6) //*prefix.litemset* is the last item of prefix
(7) $hash\_item[\_x].count{+}{+}$; $hash\_item[\_x].tid \leftarrow \cup t.tid$;
(8) **ENDFOR**
(9) **FOR** each key *c* in *hash_item* **DO**
(10) **IF** $hash\_item[c].count < min\_sup$ **THEN** delete $hash\_item[c]$;
(11) **ENDFOR**
(12) **FOR** each key *c* in *hash_item* **DO**
(13) $ASPDetector\ (c, hash\_item[c], c\_item, prefix, min\_sup, min\_rep, max\_per, max\_dis)$;
(14) **IF** *c* can be extended to *prefix* **THEN**
(15) $newprefix \leftarrow extend(c, prefix)$;
(16) $DB|_c \leftarrow Projectdatabase\ (c, DB)$;
(17) **IF** *prefix* notcontain *c_item* **THEN**
(18) delete tuples in $DB|_c$ which not contain *c_item*;
(19) **IF** $min\_sup \leq |DB|_c|$ **THEN**
(20) $AP\text{-}PrefixspanM(newprefix, DB|_c, c\_item, min\_sup, min\_rep, max\_per, max\_dis)$;
(21) **ENDFOR**

Algorithm 2: *MAPPrefixSpan* (*prefix*, *DB*, *c_item*, *min_sup*, *min_rep*, *max_per*, and *max_dis*).

pattern for short), and discover the new pattern's valid contained segment queue. The judgment of pattern growth consists of three stages: (1) potential period detection (PPD), (2) valid segment detection (VSD), and (3) valid segment mergence (VSM).

The PPD stage (as is described in steps (3)–(9) of Algorithm 3) is responsible for detecting all the possible periods of a new pattern. The hash map *hash_period* ⟨*p*, *count*⟩ (initialize *count's* value to 1) is used to record the frequency of a period. Scan the time slot queue of the parameter *occur*. Start with the first time slot and establish a sliding window of which the length is *max_per*, and calculate the time intervals between the first time slot in the sliding window and other time slots behind it. At an arbitrary time slot tid_i, calculate the interval $p_{ij} = |tid_i - tid_j|$ between tid_i and tid_j, where $i + 1 \leq j \leq \min(max\_per\text{-}i, |occur|)$. If $p_{ij} \leq max\_per$, then increase the count of the period p_{ij} of *hash_period* by one (as is described in step (4) and step (5) of Algorithm 3). Otherwise, stop calculating and slide the window to the time slot tid_{i+1} before tid_i, and repeat the calculation process. The requirement for generating a new pattern, for which the period is p_i ($1 \leq p_i \leq max\_per$), is that the frequency p_i be not less than *min_rep*; that is, $hash\_period[p_i] \geq min\_rep$.

The VSD stage is responsible for discovering all the valid contained segments of the new pattern. Firstly, for each potential period p_i, define a hash map *hash_segment* ⟨*pos*, (*rep*, *last*)⟩ to record the contained segments (as is described in step (10) of Algorithm 3), where the key *pos* is the pattern's offset, and the value (*rep*, *last*) is used to record the repeat count and the new time slot of the contained segments. At an arbitrary time slot tid_i, $tid_j\text{-}hash\_segment[pos_i].last$

Input: frequent items *fitem*, time slot table *occur*, key item *c_item*, prefix *prefix*, minimum support *min_sup*, minimum repeat count *min_rep*, maximum period *max_per*, maximum interference distance *max_dis*

(1) *hash_period* $\langle p, count \leftarrow 1\rangle$;
(2) **FOR** each tid_i in *occur.tids* **DO**
(3) **FOR** each tid_j in *occur.tids*, $i < j$ **DO**
(4) **IF**($|tid_j - tid_i|$) $\leq$ *max_per* **THEN**
(5) *hash_period* $[tid_j - tid_i]$.*count*++;
(6) **ELSE** break;
(7) **ENDFOR**
(8) **ENDFOR**
(9) **FOR** each key p_i that *hash_period* $[p_i] \geq$ *min_rep* **DO**
(10) *hash_segment* $\langle pos, (rep, last)\rangle$;
(11) *vs_set* (*patten*, *period*, *rep*, *start*) $\leftarrow \varnothing$;
(12) **FOR** each tid_i in *occur.tids* **DO**
(13) $pos_i \leftarrow tid_i$ mod p_i;
(14) **IF** (tid_i – *hash_segment*$[pos_i]$.*last*) == p_i **THEN**
(15) *hash_segment* $[pos_i]$.*rep*++;
(16) *hash_segment* $[pos_i]$.*last* $\leftarrow tid_i$;
(17) continue;
(18) **IF** *hash_segment* $[pos_i]$.*rep* $\geq$ *min_rep* **THEN**
(19) *pattern* $\leftarrow$ *extend*(*prefix*, *fitem*);
(20) *vs_set* $\leftarrow$ *vs_set* $\cup$ (*pattern*, p_i, *hash_segment*$[pos_i]$.*rep*, *hash_segment*$[pos_i]$.*last* $- p_i *$ (*hash_segment*$[pos_i]$.*rep* $- 1$));
(21) *hash_list*$[pos_i]$.*rep* $\leftarrow 1$;
(22) *hash_list*$[pos_i]$.*last* $\leftarrow tid_i$;
(23) **ENDFOR**
(24) **FOR** each key pos_i that *hash_list*$[pos_i] \geq$ *min_rep* **DO**
(25) *pattern* $\leftarrow$ *extend*(*prefix*, *fitem*);
(26) *vs_set* $\leftarrow$ *vs_set* $\cup$ (*pattern*, p_i, *hash_list*$[pos_i]$.*rep*, *hash_list*[*posi*].*last* $- p_i *$ (*hash_list*$[pos_i]$.*rep* $- 1$));
(27) **ENDFOR**
(28) *svs_set* $\leftarrow$ *sort*(*vs_set*);
(29) **FOR** each vs_i in *svs_set* **DO**
(30) **IF** (vs_i.*rep* $\geq$ *min_sup*) && (*c_item* $\in$ *pattern*) **THEN**
(31) *Output*(vs_i);
(32) *MergeSeg*(vs_i, *svs_set*, vs_i.*rep*, *min_sup*, *max_dis*);
(33) **ENDFOR**
(34) **ENDFOR**

Algorithm 3: *ASPDetector* (*fitem*, *occur*, *c_item*, *prefix*, *min_sup*, *min_rep*, *max_per*, and *max_dis*).

($pos_i = tid_i \% p_i$) is the time interval of two simultaneous occurrences. If tid_i, tid_j-*hash_segment*$[pos_i]$.*last* is equal to p_i, which means that those two occurrences are consecutive in the same contained segment. *hash_segment*$[pos_i]$.*rep* records the repeat count of a contained segment. The current contained segment, for which the offset is pos_i, is valid when *hash_segment*$[pos_i]$.*rep* $\geq$ *min_rep*. Specifically, for each period p_i, the time slot queue should be scanned once, and, for each time slot tid_i, the offset $pos_i = tid_j \% p_i$ needs to be calculated (as described in step (13) of Algorithm 3). If tid_j-*hash_segment*$[pos_i]$.*last* is equal to p_i, the current contained segment is growing; increment *hash_segment*$[pos_i]$.*rep* by one, update *hash_segment*$[pos_i]$.*last* to tid_i (as described in steps (15) and (16) of Algorithm 3), and then process the next time slot. Otherwise, the current contained segment is interrupted; at that time, if *hash_segment*$[pos_i]$.*rep* $\geq$ *min_rep*, the current contained segment is valid. It would be recorded as a tetrad in *vs_set* (as described in steps (18)–(20) of Algorithm 3). After the interruption, *hash_segment*$[pos_i]$.*rep* would be reset to 1, and *hash_segment*$[pos_i]$.*last* would be updated to tid_i (as described in steps (21) and (22) of Algorithm 3). After the time slot queue is detected, *hash_segment* would be detected again to judge whether a valid contained segment exists (as described in steps (24)–(27) of Algorithm 3). Sort all valid contained segments for which the period is p_i in ascending order by their start time slot (as described in step (28) of Algorithm 3). For each contained segment, call the function *MergeSeg* to merge all contained segments, and generate a valid contained segment queue, for which the period is p_i; start with the current contained segment (as described in steps (28)–(31) of Algorithm 3), then print the valid contained segment queue for which the key items are included.

The VSM stage is responsible for generating the valid contained segment queue. Function *MergeSeg* adopts the depth-first enumeration method to merge all possible contained segments into the queue. Generate a different contained segment queue with different initial contained segments, and, for each contained segment, use the divide-and-conquer strategy to find the segment that can be merged with the initial segments.

Input: prefix segments *prefix_segs*, contain segments set *svs_set*, repeat count of prefix segments *prefix_sum*, minimum support *min_sup*, maximum interference distance *max_dis*
(1) $VSQ \leftarrow \varnothing$;
(2) **FOR** each vs_i in *svs_set* $i > 1$ **DO**
(3) $tail \leftarrow vs_1.start + vs_1.period * (vs_1.rep - 1)$;
(4) **IF** $((vs_i.start - tail) > max\_dis * p)$ **THEN** break;
(5) **ELSE IF** $(tail > vs_i.start)$ **THEN** continue;
(6) **ELSE** $VSQ \leftarrow VSQ \cup vs_i$;
(7) **ENDFOR**
(8) **FOR** each vs_i in VSQ **DO**
(9) $newprefix\_segs \leftarrow prefix\_segs \cup vs_i$;
(10) $newprefix\_sum \leftarrow prefix\_sum + vs_i.rep$;
(11) $newsvs\_set \leftarrow svs\_set$ delete segments before vs_i;
(12) **IF** $(newprefix\_sum \geq min\_sup)$ && $(c\_item \in pattern)$ **THEN**
(13) *Output*(*newprefix_segs*);
(14) *MergeSeg* (*newprefix_segs*, *newsvs_set*, *prefix_sum*, *min_sup*, *max_dis*);
(15) **ENDFOR**

Algorithm 4: *MergeSeg* (*prefix_segs*, *svs_set*, *prefix_sum*, *min_sup*, and *max_dis*).

Input: parameters and their meaning are shown in Table 5
Output: simulation data set
(1) **FOR** $i = 1$ to C *do*
(2) $p.period$ = GaussianDist(P, $MAXPER$);
(3) $p.pattern$ = GenPattern($p.period$, T, I);
(4) $position$ = UniformDist($N/5$);
(5) **WHILE** (1) **DO**
(6) $p.local$ = GeometricDist(R, MAX = N/P);
(7) **IF** (N-$position$ < $p.local$ * $p.period$) **THEN** break;
(8) **FOR** $j = 1$ to $p.local$ **DO**
(9) $position$+=$p.period$;
(10) insert $p.pattern$ into $TIDS[position]$;
(11) **ENDFOR**
(12) $p.disturbance$ = $p.period$ * GeometricDist(D, MAXDIS);
(13) $postion$+=$p.disturbance$;
(14) **ENDWHILE**
(15) **ENDFOR**
(16) **FOR** $i = 1$ to N **DO**
(17) t = PoissonDist(I, MAX = N);
(18) $k = t - |TIDS[i]|$;
(19) random generate k symbols with $TIDS[i]$;
(20) **ENDFOR**

Algorithm 5: *DataGenerator* (N, S, $High$, $Medium$, Low, C, P, T, I, R, and D).

Then, regard these segments as the initial segments to find other segments that can be merged recursively. Repeat this process until no segments can be merged. Specifically, if the repeat count of an initial segment is not less than *min_sup*, print this segment (as described in steps (29) and (30) of Algorithm 3). For any initial segment, scan *svs_set* once and find the segment set into which the segment can be merged directly (as described in steps (2)–(7) of Algorithm 4). For the segments of such set, call the function *MergeSeg* recursively (as described in step (14) of Algorithm 4) to generate a smaller segment set (as described in step (11) of Algorithm 4), and various longer segment queues can be generated (as described in step (9) of Algorithm 4). Once the sum of the segments' repeat counts is not less than *min_sup* and key items are included in the pattern while merging, print that segment queue (as described in steps (12) and (13) of Algorithm 4).

6. Experimental Analysis

This paper first proposes an asynchronous periodic sequential pattern mining model, and, to confirm the validity and stability of the algorithm, we also design a synthetic dataset generation algorithm, which is shown in Algorithm 5.

At first, C possible asynchronous periodic sequential patterns would be generated by the algorithm, and their periodic lengths are generated by a P-expectation normal distribution.

Table 5: Parameters of synthetic dataset generation algorithm.

Parameters	Meaning	Value
N	Scale of temporal symbolic sequences	5000
S	Total number of items	300
High	Proportion of high frequency items	1/3 (60%)
Medium	Proportion of medium frequency items	1/3 (30%)
Low	Proportion of low frequency items	1/3 (10%)
C	Possible number of asynchronous periodic sequential patterns	5
P	Average periodic length of sequential patterns	5
T	Average number of item sets of sequential patterns	8
I	Average number of items	2
R	Average repeat count of contained segments	25
D	Average ratio of random disturbance to period	8

Table 6: Experimental results (length : number).

Support	Length	Period	Length of valid contained segment queue
0.3	(1 : 94), (2 : 553), (3 : 2132) (4 : 4994), (5 : 7306), (6 : 6567) (7 : 3330), (8 : 729)	(5 : 25705)	(1 : 1), (2 : 5) (3 : 18526), (5 : 2) (4 : 7170), (6 : 1)
0.35	(1 : 61), (2 : 380), (3 : 1510) (4 : 3567), (5 : 5178), (6 : 4572) (7 : 2268), (8 : 486)	(5 : 18022)	(2 : 3), (3 : 10849) (4 : 7169), (5 : 1)
0.4	(1 : 37), (2 : 248), (3 : 1014) (4 : 2544), (5 : 4014), (6 : 3888) (7 : 2106), (8 : 486)	(5 : 14337)	(2 : 2), (3 : 7168) (4 : 7167)
0.5	(1 : 1)	(5 : 1)	(3 : 1)
0.5 |0.5 |0.1	(1 : 5927), (2 : 2902)	(4 : 636) (5 : 2209) (8 : 126) (10 : 4278) (12 : 251) (15 : 735) (16 : 108) (20 : 486)	(1 : 27), (2 : 162) (3 : 572), (4 : 921) (5 : 962), (6 : 914) (7 : 858), (8 : 870) (9 : 924), (10 : 909) (11 : 723), (12 : 453) (13 : 264), (14 : 162) (15 : 81), (16 : 24) (17 : 3)
0.4 |0.3 |0.2	(1 : 59), (2 : 265), (3 : 1018) (4 : 2544), (5 : 4014), (6 : 3888) (7 : 2106), (8 : 486)	(5 : 14377) (10 : 3)	(2 : 8), (3 : 7202) (4 : 7170)

The average number of items is generated by a T-expectation normal distribution. The appearance probability consists of three levels, *High*, *Medium*, and *Low*, and the starting position of patterns is determined by an (N/5)-expectation normal distribution. Then, for each asynchronous periodic sequential pattern, a set of contained segments is generated until the end of the time slot is reached. Also, according to the contained segments, the pattern would be inserted into the corresponding time slot. The repeat counts of the contained segments are generated by an R-expectation normal distribution. Finally, scan this time slot; some items will be compensated for in the sequences, and the total number of items is generated by a $(T * I)$-expectation Poisson distribution.

A synthetic dataset is generated by the dataset generation algorithm, and this algorithm is as shown in Table 5. The *AP-PrefixspanM* algorithm is used to mine asynchronous periodic sequential patterns of the synthetic dataset. The maximum period is 20, λ is 0.2, and η is 10.

The mining results of the *AP-PrefixspanM* algorithm are shown in Table 6. When the minimum support is between 0.2 and 0.4, the mining results contain 5 asynchronous periodic sequential patterns that are preset in the dataset generation algorithm. The insert mode of the largest valid contained segment queue corresponds to these 5 sequential patterns, which means that the *AP-PrefixspanM* algorithm can precisely discover asynchronous periodic sequential patterns and their valid contained segment queue.

Figure 4 shows the distribution of lengths of asynchronous periodic sequential patterns with different minimum supports. The results show that the scale and number of sequential patterns will increase when the minimum support increases. There will only be one length-1 asynchronous

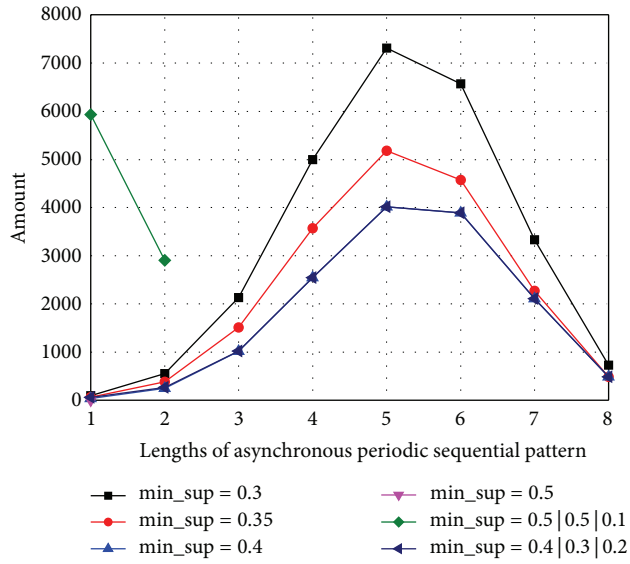

Figure 4: Distribution of lengths of asynchronous periodic sequential patterns.

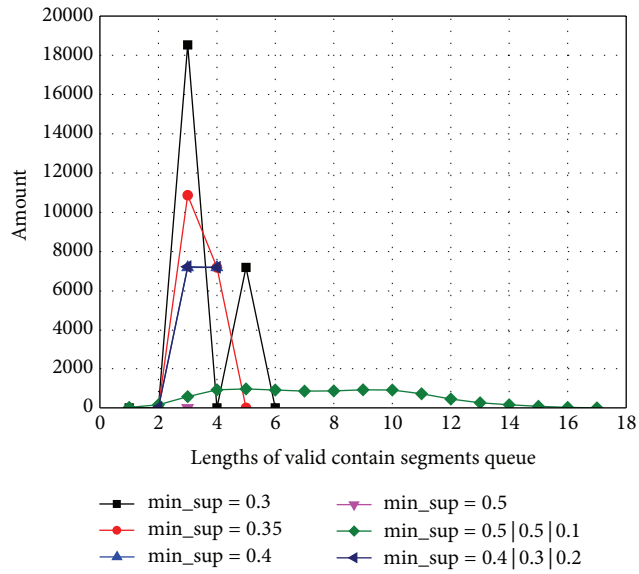

Figure 6: Distribution of valid contained segment queues of asynchronous periodic sequential patterns.

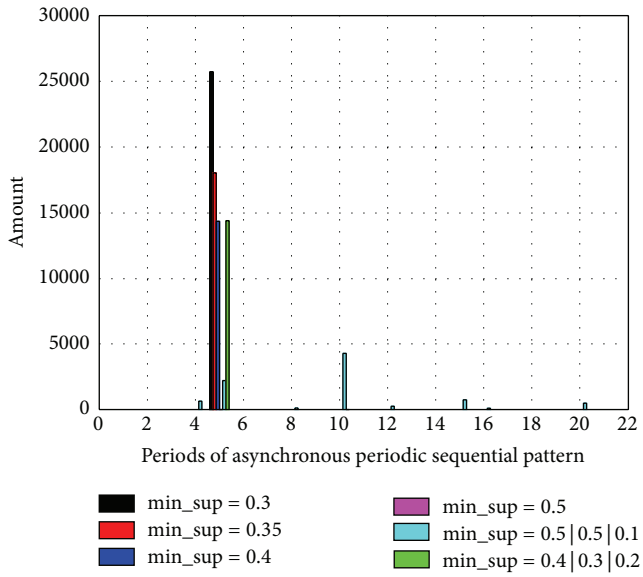

Figure 5: Distribution of periods of asynchronous periodic sequential patterns.

periodic sequential pattern when the minimum support reaches 5%. When the minimum supports of high frequency items and medium frequency items are set very high, such as 5%, but the minimum supports of low frequency items are set very low, such as 1%, a myriad of length-1 or length-2 asynchronous periodic sequential patterns will appear, which suggests that even low frequency items are focused on. Many meaningless sequential patterns would still appear if the minimum support was set very low, which means that the *AP-PrefixspanM* algorithm can effectively mine the asynchronous periodic sequential patterns with sparse items and also can avoid the generation of many meaningless patterns.

Figure 5 shows the distribution of periodic lengths of asynchronous periodic sequential patterns with different minimum supports. The results show that, except for when the minimum supports of high, medium, and low items are set as 5%, 5%, and 1%, the periodic lengths of asynchronous periodic sequential patterns are all nearly 5, which corresponds to the preset parameter $P = 5$. However, under the condition that the minimum supports of high, medium, and low items are set as 5%, 5%, and 1%, periodic lengths are random, which means that there are a great number of meaningless sequential patterns.

Figure 6 shows the distribution of lengths of valid contained segment queues of asynchronous periodic sequential patterns with different minimum supports. The results show that, except for when the minimum supports of high, medium, and low items are set as 5%, 5%, and 1%, the lengths of most valid contained segment queues are 3 or 4. Although the preset parameter R of the dataset generation algorithm is 25, because of the independent insertion and the random supplement of sequential patterns, the repeat counts of many short sequential patterns are larger than 25; therefore, the lengths of most valid contained segment queues are 3 or 4.

These experiments show that the *AP-PrefixspanM* algorithm is stable and efficient for mining asynchronous periodic sequential patterns.

7. Conclusions

In this paper, an asynchronous periodic sequential pattern mining model was proposed to discover sequential patterns that not only occur frequently but also appear periodically and to recognize the time range of their occurrences. Based on this mining model, we further propose a pattern-growth

mining algorithm named the *AP-PrefixspanM* algorithm to mine asynchronous periodic sequential patterns with multiple minimum item supports. This algorithm applies a divide-and-conquer strategy to divide the problem of mining asynchronous periodic sequential patterns into a series of mutually disjoint subproblems progressively and then to mine the patterns in such subdatabases. During the process of dividing the database, growing asynchronous periodic sequential patterns and their valid contained segment queues are generated. This is exactly what the algorithm targets. Experimental results show the efficiency and stability of the algorithm. The data which can be applied in this algorithm are those regular and frequent happening data, such as entity movement trajectory data. The algorithm can mine the regular pattern of entity movement trajectory data and predict the future movement.

The next work is to extend the *AP-PrefixspanM* algorithm and make it possible to mine asynchronous periodic spatiotemporal sequential patterns in spatiotemporal sequential databases with multiple minimum item supports.

Conflict of Interests

The authors declare that no conflict of interests exists.

Acknowledgments

This work is supported by and National Natural Science Foundation of China under Grants 61173144, 61370215, and 61370211 and National Key Technology R&D Program under Grants 2012BAH37B01 and 2012BAH45B01.

References

[1] J. Pei, J. Han, B. Mortazavi-Asl et al., "Mining sequential patterns by pattern-growth: the prefixspan approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 11, pp. 1424–1440, 2004.

[2] M. J. Zaki, "SPADE: an efficient algorithm for mining frequent sequences," *Machine Learning*, vol. 42, no. 1-2, pp. 31–60, 2001.

[3] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 55–86, 2007.

[4] F. Giannotti, M. Nanni, and D. Pedreschi, "Efficient mining of temporally annotated sequences," in *Proceedings of the 6th SIAM International Conference on Data Mining*, pp. 348–359, Bethesda, Md, USA, April 2006.

[5] B. Ozden, S. Ramaswamy, and A. Silberschatz, "Cyclic association rules," in *Proceedings of the 14th International Conference on Data Engineering (ICDE '98)*, pp. 412–421, February 1998.

[6] J. Han, W. Gong, and Y. Yin, "Mining segement-wise periodic pattern in time related databases," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pp. 214–218, 1998.

[7] J. Yang, W. Wang, and P. S. Yu, "Mining asynchronous periodic patterns in time series data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 613–628, 2003.

[8] J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," in *Proceedings of the 15th International Conference on Data Engineering (ICDE '99)*, pp. 106–115, March 1999.

[9] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, "Periodicity detection in time series databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 7, pp. 875–887, 2005.

[10] C. Sheng, W. H. Mong, and L. Lee, "Mining dense periodic patterns in time series data," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE '06)*, pp. 1–3, April 2006.

[11] W. Wang, J. Yang, and P. S. Yu, "Mining patterns in long sequential data with noise," *ACM SIGKDD Explorations Newsletter*, vol. 2, no. 2, pp. 28–33, 2000.

[12] K.-Y. Huang and C.-H. Chang, "SMCA: a general model for mining asynchronous periodic patterns in temporal databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 774–785, 2005.

[13] H. Cao, N. Mamoulis, and D. W. Cheung, "Discovery of periodic patterns in spatiotemporal sequences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 4, pp. 453–467, 2007.

[14] K. Amphawan, A. Surarerks, and P. Lenca, "Mining periodic-frequent itemsets with approximate periodicity using interval transaction-ids list tree," in *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (WKDD '10)*, pp. 245–248, January 2010.

[15] R. U. Kiran and P. K. Reddy, "Towards efficient mining of periodic-frequent patterns in transactional databases," in *Proceedings of the 21st International Conference on Database and Expert Systems Applications: Part II*, pp. 194–208, 2010.

[16] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, 2014.

[17] J. Huang, P. Zhang, X. Huangfu, and H. Sun, "A trajectory prediction approach for mobile objects by combining semantic features," *Journal of Computer Research and Development*, vol. 51, no. 1, pp. 76–87, 2014.

[18] L. Gao, J.-Y. Yang, and G.-M. Qin, "Methods for pattern mining in dynamic networks and applications," *Journal of Software*, vol. 24, no. 9, pp. 2042–2061, 2013.

[19] S. Xu and D. Pi, "Mining periodic-frequent patterns of moving objects," *Journal of Chinese Computer Systems*, vol. 35, no. 8, pp. 1705–1710, 2014.