Expert Systems with Applications 40 (2013) 4232–4240

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

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

Projection-based partial periodic pattern mining for event sequences

Kung-Jiuan Yang [a], Tzung-Pei Hong [b,c,*], Yuh-Min Chen [a], Guo-Cheng Lan [d]

[a] Institute of Manufacturing Information and Systems, National Cheng Kung University, Tainan 701, Taiwan
[b] Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan
[c] Department of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung 804, Taiwan
[d] Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan

ARTICLE INFO

Keywords:
Data mining
Encoding
Sequential pattern
Partial periodic pattern
Projection

ABSTRACT

Partial periodic pattern mining is one of the important issues in the field of data mining due to its practical applications. A partial periodic pattern consists of some periodic and non-periodic events in a specific period length, and is repeated with high frequency in an event sequence. In the past, a max-subpattern hit set algorithm was developed to discover partial periodic patterns, but its drawback is spending a large amount of time in calculating frequency counts from the redundant candidate nodes. In this study, we thus adopt an efficient encoding strategy to speed up the efficiency of processing period segments in an event sequence, and combined with the projection method to quickly find the partial periodic patterns in the recursive process. Finally, the experimental results show the superior performance of the proposed approach.

1. Introduction

Data mining techniques intended to discover sequential patterns in time series have recently been widely studied. The sequential pattern mining problem was first introduced by Agrawal, Faloutsos, and Swami (1993) and Agrawal and Srikant (1994), which was proposed to discover frequent subsequences as patterns in a sequence database. Since most of the previously developed sequential pattern mining methods followed the methodology of Apriori which had the expensive candidate generation and test problem (Agrawal & Srikant, 1995; Han, Gong, & Yin, 1998; Kim, Lim, Ng, & Shim, 2007; Masseglia, Poncelet, & Teisseire, 2009). To resolve the problem, a novel sequential pattern mining algorithm, namely *PrefixSpan*, was proposed by Pei et al. which examined only prefix subsequences and projected only corresponding postfix subsequences into a set of smaller projected databases (Pei et al., 2004). It could reduce the effort of candidate subsequence generation while mining a complete set of patterns, and substantially reduced the sizes of projected databases, thus leading to efficient processing.

In the field of sequential pattern mining, one of the important issues is finding periodic patterns, which can be categorized into two types. One is full periodic pattern mining, which considers every point in the period contributes to the cyclic behavior of the time series (Elfeky, Aref, & Elmagarmid, 2004; Faloutsos, Ranganathan, & Manolopoulos, 1994; Kim et al., 2007). It was first introduced by Özden et al. with an efficient algorithm proposed under the strict limitation of perfect periodicity in every cycle (Özden, Ramaswamy, & Silberschatz, 1998). For example, "Bill goes to the library every day." is a full periodic pattern. All the days in the year precisely contribute to the cycle pattern. However, not every activity in the real world has full regularity. It thus generates another kind of periodic pattern mining called partial periodic pattern mining, which considers most but not all points in the period contribute to the approximate cyclic behavior of the time series. The main concept of the partial periodic pattern mining was first introduced by Han et al. (1998), in which partial periodicity could be associated with a subset of the time points of the periodic behavior. In other words, it was the mixture of periodic events and non-periodic events in the same period. For example, "Gasoline prices go up on Friday." is partial periodicity because it only considers Friday and says nothing about the price fluctuations throughout the rest of the week. However, the pattern may not be found by using the full periodic pattern mining techniques since a full periodic pattern has to occur every day. To find partial periodic patterns, the period length needs to be explicitly specified. Therefore, Han et al. continually proposed a partial periodic mining algorithm called a *max-subpattern hit set* (Han, Dong, & Yin, 1999), which created a max-subpattern tree additionally as a pattern structure to store the hit counts and to represent candidate frequent patterns. Although the max-subpattern tree was essential for the derivation of frequent partial patterns and corresponding counts, the traversing of the complex tree nodes and links was time consuming.

* Corresponding author at: Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan, ROC. Tel.: +886 7 5919191; fax: +886 7 5919514

E-mail addresses: kjoyang@gmail.com (K.-J. Yang), tphong@nuk.edu.tw (T.-P. Hong), ymchen@mail.ncku.edu.tw (Y.-M. Chen), rrfoheiay@gmail.com(G.-C. Lan).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.01.021

Therefore, an algorithm which can efficiently find the partial periodic patterns without spending too much time would be desired.

Based on the above reasons, this study presents an efficient algorithm, namely projection-based partial periodic patterns algorithm (abbreviated as *PPA*), for mining partial periodic patterns with a specific period length in an event sequence. Different from the existing approaches, a strategy is designed to encode the events into event tuples and segment the event sequence to a set of properly processed period sequences. With the help of the encoding strategy, the partial periodic patterns can efficiently be explored by the proposed *PPA* algorithm from the set of period segments. Finally, the experimental results also reveal the performance of the proposed *PPA* algorithm outperforms the *max-subpattern hit set* algorithm and a baseline *Apriori*-based algorithm on synthetic datasets and a real oil price dataset.

The remaining parts of this paper are organized as follows. The related works are reviewed in Section 2. The problem to be solved and the related definitions are described in Section 3. The detail steps of the proposed *PPA* algorithm with a specific period length are described in Section 4. An example is given to illustrate the execution of the proposed *PPA* algorithm in Section 5. The experimental results on synthetic datasets and a real dataset are then revealed in Section 6, while conclusions and suggestions for future work are given in Section 7.

2. Review of related work

In this section, some related studies on sequential pattern mining and periodic pattern mining are briefly reviewed.

2.1. Sequential pattern mining

The sequential pattern mining, which lists a set of transactions with time as a time-series data in the occurring time order of the transactions, is one of important research issues in temporal data mining due to its practical applications, such as trajectory behavior analysis (Qiao, Li, Peng, & Qiu, 2010), web usage applications (Vlachos, Meek, Vagena, & Gunopulos, 2004) or uncertain sequence data models (Zhao, Yan, & Ng, 2012). In the early work of data mining, inducing association rules from a set of transaction data was often the objective due to the relationship analysis of events in databases. To effectively find sequential patterns from a set of sequence data, Agrawal et al. proposed several algorithms, such as *AprioriAll*, *AprioriSome* and *DynamicSome* (Agrawal & Srikant, 1995). Since these algorithms were based on the *Apriori*-based method, they had the expensive candidate generation and test problem. Afterward, several algorithms for sequential pattern mining were published to improve efficiency while dealing with a large set of data, such as *GSP* (Qiao et al., 2010) and *PrefixSpan* (Özden et al., 1998). The *GSP* algorithm was proposed by Srikant and Agrawal (1996) considered the time gap between items within a predefined time sliding window in the algorithm. Similarly, the *GSP* algorithm was still based on the *Apriori* algorithm. Different from the *Apriori*-based related algorithms, the *PrefixSpan* algorithm (Özden et al., 1998) was a pattern-growth technique for mining sequential patterns to find sequential patterns by prefix projections. In particular, the concept of the projection technique used in the *PrefixSpan* algorithm was derived from the similar concept of a database query in which a query condition was given by users and then the records in a database satisfying the query condition was effectively found and output. By using the projection technique, a set of projection database could continually be divided for mining when the events in a prefix subsequence were increased. Then, the search space for subsequences could effectively be reduced, and the efficiency for finding sequential patterns could thus be improved.

It is observed that the advantages of projection-based algorithms had been effectively applied in sequential pattern mining, but specialized in partial periodic pattern mining is never been discussed. Since partial periodic pattern mining is more popular in business applications, it motivates us to propose an effective method to use projection-based algorithm in mining partial periodic patterns. The supportive discussion will be made in the following section.

2.2. Periodic pattern mining

Different from traditional sequential pattern mining, periodic pattern mining could be applied to analyze and find periodic patterns in an event sequence. The periodic pattern mining could be divided into full periodic pattern mining (Özden et al., 1998; Srikant & Agrawal, 1996; Wang & Han, 2004) and partial periodic pattern mining (Han et al., 1999, 1998). For full periodic pattern mining, each position in a pattern has to be definite and nonempty. For example, assume an event sequence "*ababbbabacab*" is given, and a specified period length and a minimum support threshold are set at 2% and 70%, respectively. First, since the specified period length is 2, the sequence is split into six period sequences, "*ab*", "*ab*", "*bb*", "*ab*", "*ac*" and "*ab*". In addition, the subsequence "*ab*" appears at the first, the second, the fourth, and the sixth period sequences. Its support is then calculated as 4/6, which is 66.67%. In this example, "*ab*" is thus not a full periodic 2-pattern with the period length of 2.

As described above, this is a quite strict constraint in full periodic pattern mining since all the events in a pattern must appear in the order and at the same time in the segments with a high ratio. However, the counts of some periodic patterns in an event sequence might be a little less than the minimum support threshold, but could not be found by using the full periodic pattern mining techniques. Hence, an extension of full periodic pattern mining, namely partial periodic pattern mining was proposed to deal with the problem (Han et al., 1998). Unlike the full periodic pattern mining only dealing with definite events, a partial periodic pattern with a specified period length could include some indefinite events that were represented by the symbol "*". Continuing the previous example, take the 2-subsequence "a^*" with the period length of 2 in the six segments as an example. The count of the 2-subsequence is found to be 5. Then, its support could be calculated as 5/6, which is 83.33%. Here assume the minimum support threshold is set at 70%. The "a^*" is then a partial periodic pattern since its support satisfies the minimum support threshold. In this example, other than "a^*", the subsequence "$^*b$" is also a partial periodic pattern. As the example describes, the partial periodic pattern mining is more flexible than the full periodic pattern mining.

To find partial periodic patterns in an event sequence, Han et al. proposed an effective tree-based mining algorithm, namely *max-subpattern hit set* (Han et al., 1999). It required two scans of an event sequence to build the max-subpattern tree. In its first scan of the event sequence, all frequent 1-patterns, in each of which there is only one definite symbol and the other symbols are all "*", found from the sequence, and then a candidate max-pattern (C_{max}) is generated from the set of frequent 1-patterns as the root of the tree. For example, assume the frequent 1-pattern set includes the four patterns {a^{**}, b^{**} ,$^*b^*$, $^{**}c$}. C_{max} is then formed as {*ab*}*bc*. Notice that a position in the candidate max-pattern may be allowed to have a disjunction of more than one non-* letter with brackets, such as {*ab*} in the first position of C_{max}. Next, the second scan of the event sequence was conducted to intersect each period segment recursively with C_{max} for expanding a child node as a subpattern of C_{max} with one non-* letter replaced with the symbol '*'.

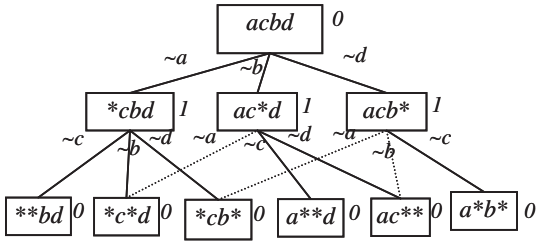

Fig. 1. An example of the max-subpattern tree.

The segments are checked with C_{max} one by one. If a segment can match a subpattern of C_{max}, a new child node for the subpattern is generated with a count 1 or the count of the existing node for the subpattern is increased by 1 (depending on whether the child node for the subpattern has existed or not). Fig. 1 shows an event sequence "*bcdedcbdacedacba*", for which the periodic length and the minimum support threshold are set at 4% and 50%, respectively. The count of each node can be accumulated from the counts of the link-ancestors (the parent nodes are with line linked) and not-linked ancestors (the parent nodes are with dotted line linked) to the root node. The reason for the not-linked ancestors (dotted line linked) is because the node cannot be recreated when it is already existed. For example, in Fig. 1 "$^*cb^*$" is formed from the node "$^*cbd$", but it is also requested to be formed from the node "acb^*". Since a node cannot duplicate, the other linking with the existing node "$^*cb^*$" will be dotted, and named it as a not-linked ancestors. These two types of ancestors will also affect the frequency count calculation. For example, the count of the new node "$^*cb^*$" is the value of 2 by its super nodes "$^*cbd$" (linked ancestor), "acb^*" (not-linked ancestor) and the counts of the two nodes. Hence, all partial periodic patterns in the built tree could be found. For more detail of (not-)linked ancestor creation or count calculation, refer to the study by Han et al. (1999).

Afterward, many published studies related to partial periodic pattern mining have applied the concept of the max-subpattern hit sets to various practical applications. For example, Aref, Elfeky, and Elmagarmid (2004) extended the max-subpattern hit set by introducing algorithms for incremental, on-line and merge mining of partial periodic patterns. Moreover, some new algorithms were also developed as tools to filter and discover candidate periods and then to apply the max-subpattern hit set to extract partial periodic patterns (Berberidis, Aref, Atallah, Vlahavas, & Elmagarmid, 2002; Berberidis, Vlahavas, Aref, Atallah, & Elmagarmid, 2002; Li, Ho, & Lee, 2009; Yang & L. Lee, 2004). These studies adopted the concept of the max-subpattern hit set to find partial periodic patterns in various problems, but they might encounter the efficient and accuracy problem while setting a lower minimum confidence threshold. As mentioned above, the max-subpattern tree is created by disjoining length-1 frequent patterns as a candidate root node. It means the more frequent 1-patterns; the more complicated root node could be created. Consequently results are more child nodes created and much time in traversing the tree for finding possible partial periodic patterns.

Therefore, this motivates our exploration of the issue of efficiently partial periodic pattern mining algorithm to eliminate the disadvantages of max-subpattern tree in discovering partial periodic patterns from an event sequence.

3. Problem statement and definitions

To explain the problem of partial periodic pattern mining with a specific period length clearly, assume a single event sequence is composed of n occurred events in an order of their occurring time stamp. Since partial periodic pattern mining is an extension issue of full periodic pattern mining, for partial periodic pattern mining problem to be solved, a set of terms related to full periodic pattern mining with a specific period length is first defined as follows.

Definition 1. S is a single event sequence which is composed of occurred events in an order of their occurring time stamp. Here let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of events, which may appear in different time stamps of S, but a time stamp only exists one event.

For example, $S = \langle abcdabcbadad \rangle$, where the set of events $I = \{a, b, c, d\}$. The first and second events, a and b, are appeared in the first and second time stamps in S, respectively, and all events in S are listed in an order of their occurring time stamp.

Definition 2. A pattern X is a subset of events, $X \subseteq I$. If $|X| = r$, the periodic pattern X is called an r-pattern. For example, the pattern $\langle ab \rangle$ contains two events and is thus called a 2-pattern.

Definition 3. Given a period length of l, a single event sequence S can be divided into $m(= \lceil \frac{n}{l} \rceil)$ mutually disjoint period segments. The single event sequence S can then be denoted as $S = \langle ps_1, ps_2, \ldots, ps_m \rangle$, where the period segment ps_{jk} is the kth event in the jth period segment ps_j.

For example, if the period length is set to 4, the sequence $S = \langle abcdabcbadad \rangle$ can be divided into three period segments with a period length of 4, $ps_1 = \langle abcd \rangle$, $ps_2 = \langle abcb \rangle$ and $ps_3 = \langle adad \rangle$. In addition, ps_{12} represents the second event $\langle b \rangle$ in the first period segment.

Definition 4. The support sup_{cp} of a candidate full pattern cp with the period length of l is the frequency count of the period segments with period length of l including the cp over the total number of period segments with period length of l in S. For example, the candidate pattern $\langle abcd \rangle$ only appears in the first period segment, and then its support value can be calculated as 1/3, which is 33.33%.

A minimum support threshold is given here to define a (frequent) full periodic pattern.

Definition 5. A (frequent) full periodic pattern is the one with its support value larger than or equal to the minimum support threshold. For example, when the minimum support threshold is set as 50%, the period segment $\langle abcd \rangle$ is not a full periodic pattern.

In this study, since our main goal is to find partial periodic patterns in a single event sequence, some other terms regarding partial periodic pattern mining based on full periodic pattern mining are given below. Different from full periodic pattern mining, there exists at least an indefinite event in a partial periodic pattern.

Definition 6. A start symbol * in a pattern can be represented as an indefinite event. For example, $\langle a^* \rangle$ is a partial periodic pattern, and not a full periodic pattern. Besides, since $\langle a^* \rangle$ is a pattern with period length of 2, $\langle a^* \rangle$ can be $\langle ab \rangle$ or $\langle ac \rangle$, but cannot be $\langle a \rangle$ or $\langle abc \rangle$.

Definition 7. A partial periodic pattern X is composed of a set of definite events. If $|X| = r$, the partial periodic pattern X is called an r-pattern. Actually, the number of definite events and indefinite events in a partial periodic pattern are equal to the number of events in a period segment. For example, the pattern $\langle ab^{**} \rangle$ with period length of 4 contains two definite, and is thus called a 2-pattern.

Definition 8. The support sup_{cpp} of a candidate partial periodic pattern *cpp* is the number of period segments with period length of *l* including *cpp* in *S* over the total number of period segments with period length of *l* in *S*. Continuing the example in Definition 3, since a candidate partial periodic 2-pattern $\langle ab^{**}\rangle$ appears in the first and second period segments, its support value can be calculated as 2/3, which is 66.67%.

As partial periodic pattern mining, a minimum support threshold is given here to define a (frequent) partial periodic pattern.

Definition 9. A candidate partial periodic pattern *cpp* is called a (frequent) partial periodic pattern (abbreviated as *PPP*), if its support value is larger than or equal to the minimum support threshold. For example, if the minimum support threshold is set as 50%, the 2-pattern $\langle ab^{**}\rangle$ is a partial periodic pattern with the period length of 4.

Based on the definitions above, the problem to be solved in the study is to find all partial periodic patterns with the period length of *l*, which their actual support values are larger than or equal to a predefined minimum support threshold, from a given single event sequence *S*. On the other hand, to effectively simplify the problem of partial periodic pattern mining, each event in each period segment of *S* is encoded as an event tuple with the event and its position identifier in the corresponding period segment of *S*. Here some definitions related to the rules in the encoding strategy are described below.

Definition 10. An event tuple et_{jk}, is composed of the event and its position identifier in the *j*th period segment ps_j, of *S*. Note that the position identifier for each event in a period segment is from 1 to period length. For example in *S*, when period length is set as 4, et_{23} represents in the third event of the second period segment ps_2.

Definition 11. An encoded period segment, eps_j, is composed of all event tuples in the original period segment ps_j. For example, in *S*, the first period segment can be encoded as $\langle(a,1),(b,2),(c,3),(d,4)\rangle$.

Definition 12. A tuple pattern *TP* is composed of a set of event tuples. If $|TP| = r$, the pattern *TP* is called an *r*-pattern. For example, the pattern $\langle(a,1),(b,2)\rangle$ with period length of 4 contains two event tuples, and is thus called a 2-pattern.

Definition 13. The support value sup_{ctp} of a candidate tuple pattern *ctp* is the number of encoded period segments including *ctp* over the number of encoded period segments in *S*. For example, in *S*, the pattern $\langle(a,1),(b,2)\rangle$ appears in eps_1 and eps_2, its support value can be calculated as 2/3, which is 66.67%.

Definition 14. Let λ be a pre-defined minimum support threshold. A candidate tuple *r*-pattern is called a frequent tuple pattern (abbreviated as *FTP*), if $sup_{ctp} \geqq \lambda$. For example, if the minimum support threshold is set as 50%, $\langle(a,1),(b,2)\rangle$ is a frequent tuple pattern with the period length of 4.

Actually, a frequent tuple pattern is equal to a (frequent) partial periodic pattern. For example, the frequent tuple pattern $\langle(a,1),(b,2)\rangle$ with period length of 4 can be returned to its corresponding (frequent) partial periodic pattern $\langle ab^{**}\rangle$ with period length of 4. As above describes, the problem of partial periodic pattern mining can be simplified effectively and handle efficiently by using the encoding strategy.

4. The proposed algorithm

The details of the proposed *PPA* algorithm for mining partial periodic patterns with a specific period length are described in this section. It is based on the projection technique with an effective encoding strategy in the algorithm to help its execution. The encoding strategy is first described below.

4.1. The encoding strategy for period segments

In this study, the effective encoding strategy is adopted to speed up the efficiency of processing period segments in an event sequence by using the proposed algorithm (Han et al., 1998). Based on Definitions 10–14, every event in a period segment could be encoded into an event tuple which consists of itself and its position identification number. For example, the period segment $\langle abcd\rangle$ can be encoded as $\langle(a,1),(b,2),(c,3),(d,4)\rangle$, where the first event tuple $\langle(a,1)\rangle$ represents the event *a* and its first position identifier in the period segment $\langle abcd\rangle$. In addition, the event tuple in a period segment have to be sorted first in position identification number and then in alphabetical order of them. For example, the encoded period segment $\langle(a,1),(b,2),(c,3),(a,4)\rangle$ has been sorted in position identification number and alphabetical order of the items. Through this strategy, the proposed *PPA* algorithm can effectively find partial periodic patterns in a set of encoded period segments.

4.2. The proposed mining algorithm, PPA

The procedures of the proposed *PPA* algorithm are then stated as follows.

- INPUT: An event sequence *S* with *n* events, a periodic length of interest *l*, and a minimum support threshold *min_sup*.
- OUTPUT: A final set of all (frequent) partial periodic patterns of *PPPs*.
- STEP 1: Divide the event sequence *S* with the period length of interest *l* into multiple period segments, $S = \langle ps_1, ps_2, \ldots, ps_m\rangle$.
- STEP 2: Encode each event in ps_j to an event tuple et_{jk}, with the event and its position identifier in ps_j and then generate an encoded period segment eps_j. All the encoded period segments are collected as an encoded period segment database, *EPSD*.
- STEP 3: For each *k*th tuple event et_{jk} in eps_j as candidate tuple pattern, find the support value:

$$Sup_{ctp} = \frac{fc_{ctp}}{m}$$

  where sup_{ctp} is the frequency count of the candidate tuple pattern, and *m* is the number of encoded period segments in *EPSD*.
- STEP 4: For each candidate tuple pattern (*ctp*) in *EPSD*, if its actual support value is larger than or equal to the minimum support threshold *min_sup*, put the pattern in the set of frequent tuple 1-patterns, FTP_1.
- STEP 5: Gather the event tuples appearing in the set of FTP_1, and removed the event tuples which are not in FTP_1 the *EPSD*.
- STEP 6: Set $r = 1$, where *r* represents the number of event tuples in the processed patterns.
- STEP 7: Process each event tuple *X* in the set of FTP_1 in the position number and alphabetical order of them by the following substeps.
  - (a) Find the relevant encoded period segments including *X* in *EPSD*, and put the encoded period segments in the set of projected encoded period segment database $epsd_X$ of the *X*.
  - (b) Find all the frequent tuple patterns with *X* as their prefix pattern by the *Finding-FTP*(*X*, $epsd_X$, *r*) procedure. Let the set of returned encoded tuple patterns be FTP_X.

- STEP 8: Output the set of all partial periodic patterns (*PPPs*) in the FTP_X.

All the partial periodic patterns are found by *Finding-FTP*(X, $epsd_X$, r) procedure, and the procedure is stated below.

4.3. The finding-FTP(X, $epsd_X$, r) procedure

- Input: A prefix r-pattern X and its corresponding projected encoded period segment database $epsd_X$.
- Output: The frequent event tuple patterns with the prefix pattern X, FTP_X.
- PSTEP 1: Initialize the temporary tuple pattern TTP_X table as an empty table, in which each tuple consists of two fields: pattern and the actual count of the pattern.
- PSTEP 2: For each jth period segment eps_j in $epsd_X$, do the following substeps.
  (a) Get each event tuple $\langle et_j \rangle$ located after X in ps_j.
  (b) Generate the $(r+1)$-event tuple X' composed of the prefix r-pattern X and $\langle et_j \rangle$, put X' in the TTP_X table, and add the value of 1 to its corresponding count field value in the TTP_X table.
- PSTEP 3: For each $(r+1)$-event tuple X' in the TTP_X table, do the following substeps.
  (a) Calculate the support value $sup_{X'}$ of the $(r+1)$-event tuple X' as:

$$Sup_{X'} = \frac{count_{X'}}{m}$$

where $count_{X'}$ is the count field value of X' in the TTP_X table.
  (b) If the actual support value $sup_{X'}$ of the X' is larger than or equal to the minimum support threshold λ, put X' in the set of frequent event tuple $(r+1)$-patterns, $FTP_{(r+1),X'}$.
- PSTEP 4: Set $r = r+1$, where r represents the number of event tuples in the processed patterns.
- PSTEP 5: Process each pattern X' in the set of $FTP_{r,X}$ in the position number and alphabetical order of them by the following substeps.
  (a) Find the relevant encoded period segments including X' in $epsd_X$, and put the encoded period segments in the set of projected encoded period segment database $epsd_{X'}$ of the X'.
  (b) Find all the frequent tuple patterns with X' as their prefix pattern by the *Finding-FTP*(X', $epsd_{X'}$, r) procedure. Let the set of returned frequent tuple patterns be FTP_X.
- PSTEP 6: Return the frequent tuple patterns in the set of FTP_X.

5. An example of PPA

In this section, an example is given to show how the proposed algorithm can easily be used to find all partial periodic patterns with a specific length of interest from an event sequence. Assume an event sequence "*abbdcbabdccacabdabac*" is used for mining. There are four non-duplicated events in the sequence, respectively denoted *a*–*d*. The length of the event sequence is 20. Moreover, the support threshold *min_sup* and the periodic length of interest are set as 50% and 5%, respectively. To find the partial periodic patterns within a specific periodic length from the event sequence S, the proposed *PPA* algorithm then proceeds as follows.

- STEP 1: The event sequence can be divided into four period segments with the period length of interest l (=5), such as ps_1 = "*abbdc*", ps_2 = "*babdc*", ps_3 = "*cacab*" and ps_4 = "*dabac*", respectively.
- STEP 2: The events in each period segment is encoded to an event tuple. Take the first period segment S_1 as an example. The first period segment "*abbdc*" can be encoded as the encoded period segment, $eps_1 = \langle (a,1), (b,2), (b,3), (d,4), (c,5) \rangle$ and then put it into the set of encoded period segment database, *EPSD*. The same process can be done for the other three period segments. The results for all the period segments are shown in Table 1.

Table 1
The set of *EPSD* in this example.

eps_i	Encoded period segment
1	$\langle (a,1), (b,2), (b,3), (d,4), (c,5) \rangle$
2	$\langle (b,1), (a,2), (b,3), (d,4), (c,5) \rangle$
3	$\langle (c,1), (a,2), (c,3), (a,4), (b,5) \rangle$
4	$\langle (d,1), (a,2), (b,3), (a,4), (c,5) \rangle$

- STEP 3: The support value of each possible event tuple in *EPSD* can be found. Take the event tuple $\langle (a,4) \rangle$ as an example. In Table 1, event tuple $\langle (a,4) \rangle$ appears the two period segments, eps_3 and eps_4. Then, the count value of event tuple $\langle (a,4) \rangle$ is 2. In addition, the total number of encoded period segments in Table 1 is 4. The support value of event tuple $\langle (a,4) \rangle$ can be calculated as 2/4, which 50%. The other event tuples in *EPSD* can similarly be processed in the same way. Next, each event tuple 1-pattern is checked whether or not its support value is larger than or equal to the minimum support threshold *min_sup*. If it is, the pattern can be put in the sorted set of frequent event tuple 1-patterns, FTP_1; otherwise, the pattern can be removed. In this example, since only the five 1-patterns, $\langle (a,2) \rangle$, $\langle (b,3) \rangle$, $\langle (a,4) \rangle$, $\langle (d,4) \rangle$ and $\langle (c,5) \rangle$, satisfy the minimum support threshold *min_sup*, they are put in the set of frequent event tuple 1-patterns (FTP_1), as shown in Table 2.
- STEP 4: All the event tuples appearing in the set of FTP_1 are gathered. And the infrequent event tuples are removed from *EPSD*. Therefore, for eps_1, only the event tuples, $(b,3)$, $(d,4)$ and $(c,5)$ are kept. The same process can be done for the other three encoded period segments. The results for all the encoded period segments in *EPSD* are shown in Table 3.
- STEP 5: The variable r is then set at 1, where r represents the number of event tuples in the patterns to be processed.
- STEP 6: The five 1-patterns in the set of FTP_1 are sequentially processed in the position number and alphabetical order. The sorted patterns are $\langle (a,2) \rangle$, $\langle (b,3) \rangle$, $\langle (a,4) \rangle$, $\langle (d,4) \rangle$ and $\langle (c,5) \rangle$. First, take $\langle (a,2) \rangle$ as a prefix to describe the detail *Finding-FTP* subroutine, and its projected encoded period segments $epsd_{\langle (a,2) \rangle}$ in Table 3 include the second, the third and the fourth sequences, $\langle (a,2), (b,3), (d,4), (c,5) \rangle$, $\langle (a,2), (a,4) \rangle$ and $\langle (a,2), (b,3), (a,4), (c,5) \rangle$. Note that only the event tuples located after the prefix $\langle (a,2) \rangle$ and the prefix can be kept in the projected encoded segments.

Next, all the partial periodic patterns with the prefix $\langle (a,2) \rangle$ are then found by the *Finding-FTP*(X, $epsd_X$, r) procedure with the parameters $X = \langle (a,2) \rangle$, $epsd_X = epsd_{\langle (a,2) \rangle}$, and $r = 1$. The details of the *Finding-FTP*(X, $epsd_X$, r) procedure executed on this example are described below.

- PSTEP 1: The temporary tuple pattern $TTP_{\langle (a,2) \rangle}$ table is initialized as an empty table, in which each event tuple pattern consists of two fields: event tuple pattern and the count value of the pattern.
- PSTEP 2: For each period segment in $epsd_{\langle (a,2) \rangle}$ of $\langle (a,2) \rangle$, all possible 2-patterns with the prefix $\langle (a,2) \rangle$ are generated. Take the first projected encoded period segment $\langle (a,2), (b,3), (d,4), (c,5) \rangle$ in $epsd_{\langle (a,2) \rangle}$ as an example. Since event tuples $\langle (b,3) \rangle$, $\langle (d,4) \rangle$ and $\langle (c,5) \rangle$ are located after the prefix $\langle (a,2) \rangle$ in the segment, the candidate patterns, $\langle (a,2), (b,3) \rangle$ $\langle (a,2), (d,4) \rangle$ and $\langle (a,2), (c,5) \rangle$ can be generated from the segment $\langle (a,2), (b,3),$

Table 2
The set of frequent 1-patterns in this example.

1-Pattern	Support (%)
⟨(*a*,2)⟩	75
⟨(*b*,3)⟩	75
⟨(*a*,4)⟩	50
⟨(*d*,4)⟩	50
⟨(*c*,5)⟩	75

Table 3
The modified set of *EPSD* in this example.

eps_i	Encoded period segment
1	⟨(*b*,3), (*d*,4), (*c*,5)⟩
2	⟨(*a*,2), (*b*,3), (*d*,4), (*c*,5)⟩
3	⟨(*a*,2), (*a*,4)⟩
4	⟨(*a*,2), (*b*,3), (*a*,4), (*c*,5)⟩

Table 4
The counts of all candidate 2-patterns with the prefix ⟨(*a*,2)⟩.

2-Pattern	Count
⟨(*a*,2), (*b*,3)⟩	2
⟨(*a*,2), (*d*,4)⟩	1
⟨(*a*,2), (*c*,5)⟩	2
⟨(*a*,2), (*c*,3)⟩	1
⟨(*a*,2), (*a*,4)⟩	2
⟨(*a*,2), (*b*,5)⟩	1

Table 5
The counts of all candidate 3-patterns with the prefix ⟨(*a*,2),(*b*,3)⟩.

3-Pattern	Count
⟨(*a*,2), (*b*,3), (*d*,4)⟩	1
⟨(*a*,2), (*b*,3), (*c*,5)⟩	2
⟨(*a*,2), (*b*,3), (*a*,4)⟩	1

Table 6
The final set of all partial periodic patterns in this example.

i-Pattern	Support (%)	Partial periodic pattern
⟨(*a*,2)⟩	75	⟨$*a***$⟩
⟨(*b*,3)⟩	75	⟨$**b**$⟩
⟨(*a*,4)⟩	50	⟨$***a*$⟩
⟨(*d*,4)⟩	50	⟨$***d*$⟩
⟨(*c*,5)⟩	75	⟨$****c$⟩
⟨(*a*,2), (*b*,3)⟩	50	⟨$*ab**$⟩
⟨(*a*,2), (*b*,3), (*c*,5)⟩	50	⟨$*ab*c$⟩
⟨(*a*,2), (*a*,4)⟩	50	⟨$*a*a*$⟩
⟨(*a*,2), (*c*,5)⟩	50	⟨$*a**c$⟩
⟨(*b*,3), (*d*,4)⟩	50	⟨$**bd*$⟩
⟨(*b*,3), (*d*,4), (*c*,5)⟩	50	⟨$**bdc$⟩
⟨(*b*,3), (*c*,5)⟩	50	⟨$**b*c$⟩
⟨(*d*,4), (*c*,5)⟩	50	⟨$***dc$⟩

(*d*,4), (*c*,5)⟩, and then the candidates are put in the $TTP_{\langle(a,2)\rangle}$ table of ⟨(*a*,2)⟩, and the value of 1 is added into its corresponding count field value in the $TTP_{\langle(a,2)\rangle}$ table. The same process is done for the other projected encoded period segments in $epsd_{\langle(a,2)\rangle}$. The results for the candidate 2-patterns and their counts are shown in Table 4.

- PSTEP 3: The support value of each candidate pattern in $TTP_{\langle(a,2)\rangle}$ table is calculated. The process is the same as that mentioned in STEP 3. In this example, in Table 4, since there are three candidates ⟨(*a*,2), (*b*,3)⟩, ⟨(*a*,2), (*a*,4)⟩ and ⟨(*a*,2), (*c*,5)⟩ with the counts are found to be 2, the support value can be calculated as 2/4, which is 50%. Next, the candidates are put in the set of $FTP_{2,\langle(a,2)\rangle}$ since its support value is larger than or equal to the support threshold (=50%).
- PSTEP 4: The value of the variable *r* is updated as 2.

All the partial periodic patterns with the prefix ⟨(*a*,2), (*b*,3)⟩ are then found by invoking the *Finding-FTP*(*X*, $epsd_X$, *r*) procedure again, with the parameters *X* = ⟨(*a*,2), (*b*,3)⟩, $epsd_X = epsd_{\langle(a,2),\ (b,3)\rangle}$, and *r* = 2. Take the first projected encoded segment ⟨(*a*,2), (*b*,3), (*d*,4), (*c*,5)⟩ as an example. The event tuples located after ⟨(*a*,2), (*b*,3)⟩ in the projected segment includes the two event tuples, ⟨(*d*,4)⟩ and ⟨(*c*,5)⟩. In addition, the two partial periodic 3-pattern with ⟨(*a*,2), (*b*,3)⟩ as their prefixes, ⟨(*a*,2), (*b*,3), (*d*,4)⟩ and ⟨(*a*,2), (*b*,3), (*c*,5)⟩ are generated. The same process is done for the other projected encoded segment in $epsd_{\langle(a,2),\ (b,3)\rangle}$. The results for the candidate 3-patterns and their counts are shown in Table 5. Because ⟨(*a*,2), (*b*,3), (*c*,5)⟩ with the counts are found to be 2, the support value can be calculated as 2/4, which is 50%. Next, the candidate ⟨(*a*,2), (*b*,3), (*c*,5)⟩ is put in the set of $FTP_{3,\langle(a,2)\rangle}$ since its support value is larger than or equal to the support threshold (=50%). Because there is no more event tuple after ⟨(*c*,5)⟩, the process of finding partial periodic patterns with the prefix ⟨(*a*,2), (*b*,3), (*c*,5)⟩ can be terminated.

According to the above information, the next prefix to be processed is changed to ⟨(*a*,2), (*a*,4)⟩. The above process is executed again for the prefix ⟨(*a*,2), (*a*,4)⟩. Similarly, since there are no projected segments in the $epsd_{\langle(a,2),(a,4)\rangle}$, the prefix to be processed is changed to ⟨(*a*,2), (*c*,5)⟩. And the set of the projected encoded period segments $epsd_{\langle(a,2),(c,5)\rangle}$ of the prefix ⟨(*a*,2), (*c*,5)⟩ includes the two ones, ⟨(*a*,2), (*b*,3), (*d*,4), (*c*,5)⟩ and ⟨(*a*,2), (*b*,3), (*a*,4), (*c*,5)⟩. But, since there is no event tuple after ⟨(*c*,5)⟩. Then, the process of finding partial periodic patterns with the prefix ⟨(*a*,2), (*c*,5)⟩ can be terminated. The above process is recursively executed until all the 1-patterns in FTP_1 have been done. All the partial periodic patterns in this example are then found, as shown in Table 6.

PSTEP 5: In this example, there are thirteen partial periodic patterns with the period length of 5 in Table 6 are output to users.

As this example describes, the proposed mining algorithm can efficiently generates candidate patterns with the help of the encoding strategy. The efficiency of the mining process can thus be improved.

6. Experimental evaluation

A series of experiments was conducted to compare the performance of the proposed projection-based partial periodic pattern algorithm (named *PPA*), the traditional partial periodic pattern mining algorithm (*Max-Subpattern* hit set algorithm, *MSA*) (Han et al., 1999) and the developed baseline *Apriori-based* partial periodic pattern mining algorithm (abbreviated as *APA*) under different parameter values. The experiments were implemented in Visual 2010 C# and executed on a PC with 2.4 GHz CPU and 3.0 GB memory.

6.1. Experimental datasets

To show evaluate the practical performance of the algorithms, we have performed several experiments on both IBM synthetic datasets (*S3I4D50K-500K*) IBM Quest Data Mining Projection, 1996 and a real dataset, which was collected from the Bureau of Energy, Ministry of Economic Affairs of Taiwan (http://www.moeaboe.gov.tw/oil102/), as our experimental datasets. In the experimental evaluation, the parameters used in the *IBM* data generator are *S*, *I*, *N* and *D*, which represented the average length

of events per sequence, the average length of maximal potentially frequent events, the total number of different events, and the total number of events, respectively.

On the other hand, the daily prices data for Brent crude oil was collected from January 2000 to December 2011. To observe the price change trend in the long-term time series data, a converted rule, which the difference in closing prices between it and its last day for each day was calculated to obtain the price change of the two days, was used to convert the original dataset to a price changes data. For example, the price in December 13, 2011 was 100, and the price of its last day was 97.9. Then, the difference in prices of the two days could be calculated as (100–97.9) which is 2.1. Based on the discretization rule, the event codes for Brent crude oil {*A,B,C,D,E,F,G*} uses the threshold values of {−6, −2, −0.1, 0.1, 2, 6} to represent the different level of price differences {sharp declines, small declines, little shifts, small increases, sharp increases, very high values} respectively. Finally, after the converted process, there were a total of 3125 closed prices in this dataset, and the dataset was used in the experiments to compare the performance of the algorithms and to further find potential or interesting partial periodic patterns.

6.2. Evaluation of the number of candidate patterns generated

Experiments were first made on the synthetic datasets to evaluate the difference in the number of candidate patterns generated by the three algorithms. Fig. 2 shows the comparisons in the number of candidates of the three algorithms for the synthetic datasets with various parameter settings, including *min_sup*, *D*, and *period length* (*PL*), respectively.

It could be seen in Fig. 2 that the numbers of candidate patterns generated by the proposed *PPA* algorithm were less than that generated by the *MSA* and the *APA* algorithms. The main reason for this was that *MSA* algorithm adopted the union of all 1-events to build a max-subpattern tree, many redundant candidate patterns had to be generated from the tree structure. As disadvantages of *Apriori* algorithm in association-rule mining, the baseline *APA* algorithm also had the same disadvantages including a huge number of candidates generation and multiple data scans. However, our proposed *PPA* algorithm, which was based on projection-based technique concept, could effectively reduce search space in the recursive mining process, and thus the *PPA* could avoid generating unnecessary candidate patterns in mining. In terms of number of candidate patterns, the proposed *PPA* algorithm outperformed the other two algorithms, *MSA* and *APA*.

6.3. Efficiency evaluation

Experiments were then made to evaluate the efficiency of the three algorithms, and Fig. 3 shows the execution time comparisons of the three algorithms for the synthetic datasets with the three parameter settings, *min_sup*, *D* and *period length* (*PL*), respectively.

It could be seen in the figures that the efficiency of the proposed *PPA* algorithm was better than the *MSA* and the *APA* algorithms, especially when the minimum support threshold decreased, the data size increased or the period length increased.

6.4. Evaluation on a real dataset

In this section, experiments were made on the real oil prices dataset to evaluate the execution efficiency of the three algorithms, *PPA*, *MSA* and *APA* under different period lengths. Fig. 4 shows execution efficiency of the three algorithms with *min_sup* = 10%, for the real datasets with period lengths varying from 5 to 30. In addition, Fig. 5 shows execution efficiency of the three algorithms, for the real dataset with the minimum support thresholds (*min_sup*) varying from 2% to 20% when the periodic length (*PL*) was set at 5.

As shown in these figures, it could obviously be observed that the performance of the proposed *PPA* algorithm was better than the *MSA* and the *APA* algorithms in terms of both of execution efficiency, especially when the periodic length increased. The main reason is that the *MSA* used a candidate maximal pattern (C_{max})

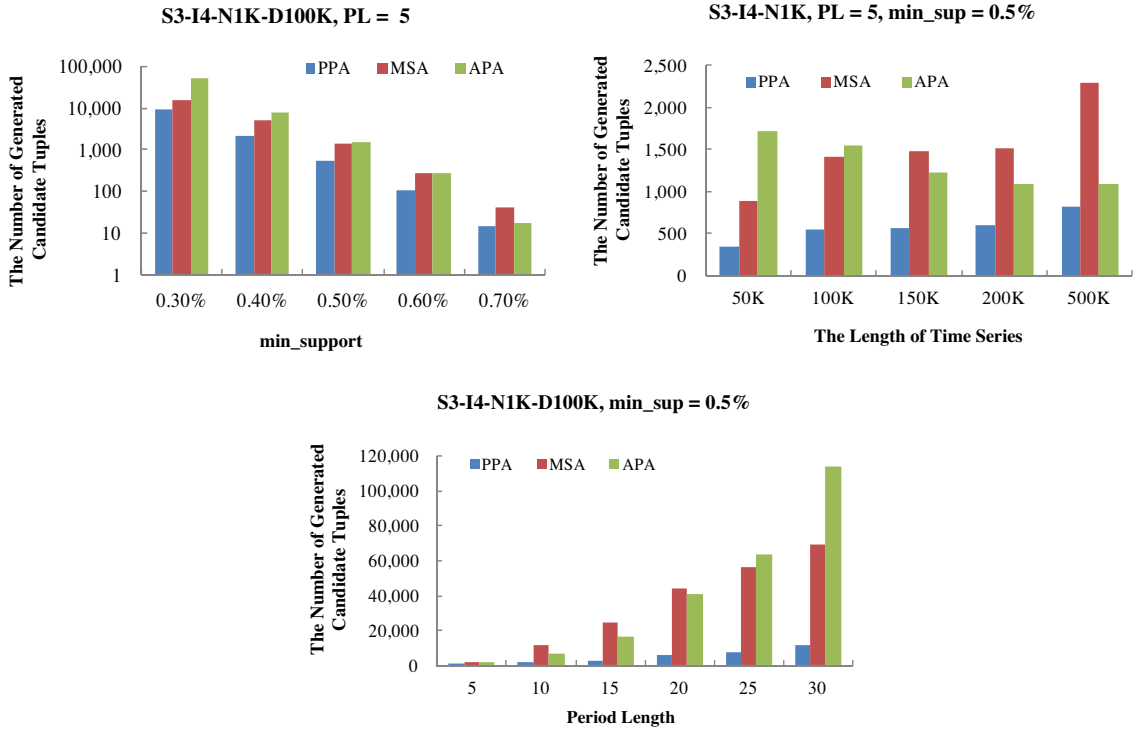

Fig. 2. Comparison of the numbers of candidate patterns generated by the three algorithms along with various parameter settings.

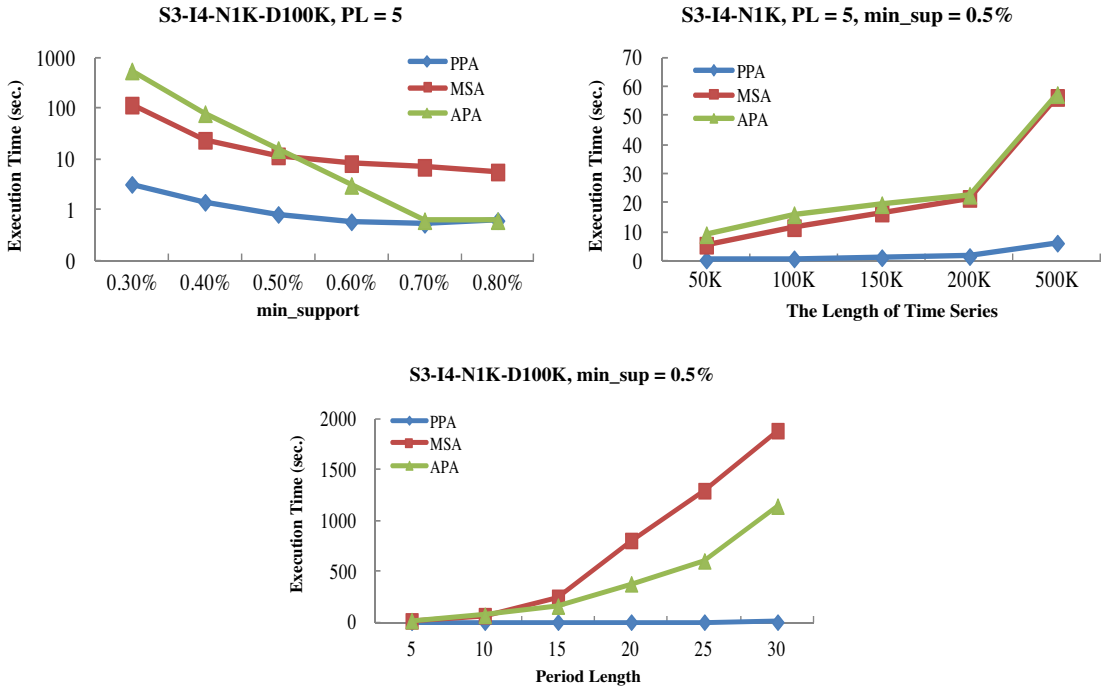

Fig. 3. Execution time of the three algorithms along with various parameter settings.

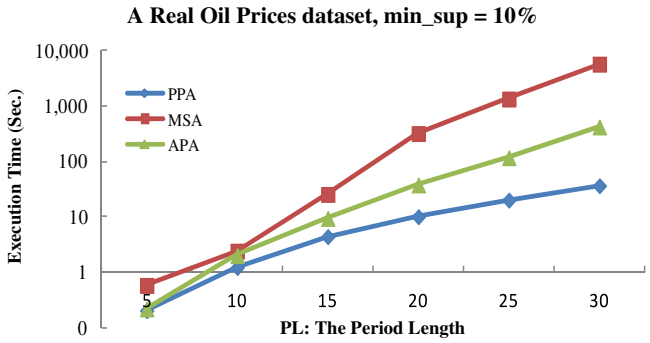

Fig. 4. Execution time of the three algorithms along with different period lengths.

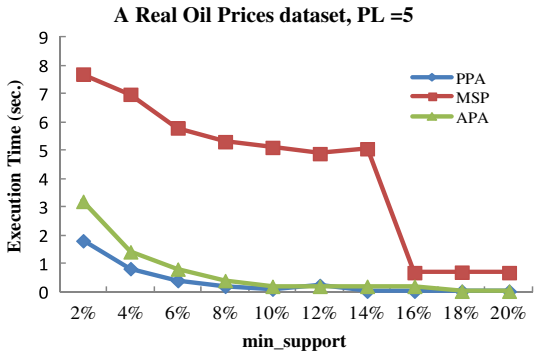

Fig. 5. Execution time of the three algorithms along with different minimum support thresholds.

that consisted of all frequent 1-patterns to build the max-subpattern tree. Hence, the *MSA* had to spend a huge amount of execution time on building the maximum sub-pattern tree and traversing the built tree to find partial periodic patterns in mining when number of frequent 1-patterns increased. Different from the *MSA* and the *APA*, the proposed *PPA* adopted the projection-based technique to effectively mining partial periodic patterns. Thus, the execution efficiency of the proposed *PPA* outperformed the *MSA* and *APA* algorithms in finding partial periodic patterns under different periodic lengths.

6.5. Pattern explanation

In the experiment, other than performance evaluation of the algorithms, several practical partial periodic patterns for the real oil prices dataset were also listed, as shown in Table 7. Note that to observe the recent oil price changes, the real dataset only involved the oil prices data within 2011, and then the length of the finally converted event sequence was 260 after the pre-processing. In addition, since a week included five days, the periodic length was set at 5. Based on above information, all the partial periodic patterns could be discovered from the real dataset (only year 2011) when the periodic length and the minimum support threshold were set at 5% and 10%, respectively. Table 7 then showed the partial results of all the discovered patterns.

As Table 7 showed, all patterns could easily be explained. Take the second partial periodic pattern in Table 7 as an example. The pattern could be explained as "the price of Brent oil of year 2011, there was 11.54% possibility for small increases (event *E*) on Wednesday and then small declines (event *C*) on Thursday". All the other patterns could be similarly explained in the same fashion.

Table 7
The partial result of the discovered patterns in the real dataset (year 2011).

Partial periodic pattern	Support (%)
⟨**EE*⟩	19.23
⟨**EC*⟩	11.54
⟨CE***⟩	11.54
⟨E*E**⟩	15.38
...	...

7. Conclusion

In this paper, we presented an efficient projection-based algorithm, namely *PPA*, to find partial periodic patterns on event sequence with a period length. Different from the maximum subpattern tree of the traditional *MSA* algorithm or the baseline *APA* algorithm, in particular, the proposed *PPA* algorithm adopts the projection-based technique can eliminate the disadvantages of the above two algorithms in dealing with the problem of partial periodic pattern mining. In addition, the encoding strategy is used to encode the events and segment the event sequence to a set of properly processed period segments. The experimental results show that the proposed *PPA* algorithm is more efficient than the other two algorithms to discover partial periodic patterns for the synthetic datasets and the real oil price dataset. Finally, in future work we will attempt to handle the maintenance problem of partial periodic pattern mining when the events are inserted, deleted or modified.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithm for mining association rules. In *Proceedings of the international conference on very large data bases* (pp. 487–499).

Agrawal, R., & Srikant, R. (1995). Mining sequential pattern. In *Proceedings of the international conference on data engineering* (pp. 3–14).

Agrawal, R., Faloutsos, C., Swami, A. (1993). Efficient similarity search in sequence databases. In *Proceedings of the fourth international conference foundations of data organization and algorithms* (pp. 69–84).

Aref, W. G., Elfeky, M. G., & Elmagarmid, A. K. (2004). Incremental, online and merge mining of partial periodic patterns in time series databases. *IEEE Transactions on Knowledge Data Engineering, 16*(3), 332–342.

Berberidis, C., Aref, W. G., Atallah, M., Vlahavas, I., & Elmagarmid, A. K. (2002). Multiple and partial periodicity mining in time series databases. In *Proceedings of the 15th European conference on artificial intelligence* (pp. 370–374).

Berberidis, C., Vlahavas, I., Aref, W. G., Atallah, M., & Elmagarmid, A. K. (2002). On the discovery of weak periodicities in large time series. *Lecture Notes in Computer Science, 2431*, 169–186.

Elfeky, M. G., Aref, W. G., & Elmagarmid, A. K. (2004). Using convolution to mine obscure periodic patterns in one pass. In *Proceedings of the 9th international conference on extending database technology* (pp. 605–620).

Faloutsos, C., Ranganathan, M., & Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. In *Proceedings of the ACM SIGMOD international conference management of data* (pp. 419–429).

Han, J., Dong, G., & Yin, Y. (1999). Efficient mining of partial periodic patterns in time series databases. In *Proceedings of the 15th international conference data engineering* (pp. 106–115).

Han, J., Gong, W., & Yin, Y. (1998). Mining segment-wise periodic patterns in time-related databases. In *Proceedings of the 4th International Conference Knowledge Discovery and Data Mining*.

IBM Quest Data Mining Projection (1996).Quest synthetic data generation code. Available from http://www.almaden.ibm.com/cs/quest/syndata.htm.

Kim, C., Lim, J., Ng, R. T., & Shim, K. (2007). SQUIRE: Sequential pattern mining with quantities. *The Journal of Systems and Software, 80*(10), 1726–1745.

Li, H. F., Ho, C. C., & Lee, S. Y. (2009). Incremental updates of closed frequent itemsets over continuous data streams. *Expert Systems with Applications, 36*(2), 2451–2458.

Masseglia, F., Poncelet, P., & Teisseire, M. (2009). Efficient mining of sequential patterns with time constraints: Reducing the combinations. *Expert Systems with Applications, 36*(2), 2677–2690.

Özden, B., Ramaswamy, S., & Silberschatz, A. (1998). Cyclic association rules. In *Proceedings of the 14th international conference data engineering* (pp. 412–421).

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., et al. (2004). Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge Data Engineering, 16*(11), 1424–1440.

Qiao, S., Li, T., Peng, J., & Qiu, J. (2010). Parallel sequential pattern mining of massive trajectory data. *International Journal of Computational Intelligence Systems, 3*(3), 343–356.

Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th international conference extending database technology* (pp. 3–17).

Vlachos, M., Meek, C., Vagena, Z., & Gunopulos, D. (2004). Identifying similarities, periodicities and bursts for online search queries. In *Proceedings of the ACM SIGMOD international conference on management of data* (pp. 131–142).

Wang, J., & Han, J. (2004). BIDE: Efficient mining of frequent closed sequences. In *Proceedings of the 20th international conference on data engineering* (pp. 79–90).

Yang, W., & Lee, G.L. (2004). Efficient partial multiple periodic patterns mining without redundant rules. In *Proceedings of the IEEE annual international computer software and applications conference* (Vol. 1, pp. 430–435).

Zhao, Z., Yan, D., & Ng, W. (2012). Mining probabilistically frequent sequential patterns in uncertain databases. In *Proceedings of the 15th international conference on extending database technology* (pp. 74–85).