Efficient Mining of Frequent Episodes from Complex Sequences

Kuo-Yu Huang and Chia-Hui Chang
Department of Computer Science and Information Engineering,
National Central University, Chung-Li, Taiwan 320
want@db.csie.ncu.edu.tw, chia@csie.ncu.edu.tw

Abstract

Discovering patterns with highly significance is an important problem in data mining discipline. An episode is defined to be a partially ordered set of events for a consecutive and fixed time intervals in a sequence. Previous studies in episodes consider only frequent episodes in a sequence of events (called simple sequence). In real world, we may find a set of events at each time slot in terms of various intervals (hours, days, weeks, etc.) We refer to such sequences as complex sequences. Mining frequent episodes in complex sequences has more extensive applications than in simple sequences. In this paper, we discuss the problem on mining frequent episodes in a complex sequence. We extend previous algorithm MINEPI to MINEPI+ for episode mining from complex sequences. Furthermore, a memory-anchored algorithm called EMMA is introduced for the mining task. Experimental evaluation on both real world and synthetic data sets shows that EMMA is more efficient than MINEPI+.

1. Introduction

Most data mining and machine learning techniques are adapted towards the analysis of unordered collections of data, e.g. transaction databases and sequence databases [1, 3, 7, 8, 26, 27, 30]. However, there are important application areas where the data to be analyzed is ordered, e.g. alarms in a telecommunication network, user interface actions, occurrences or recurrent illnesses, etc. One basis problem in analyzing such a sequence is to find frequent *episodes*, i.e. collections of events occurring frequently together [20, 19, 21]. The goal of episode mining is to find relationships between events. Such relationships can then be used in an on-line analysis to better explain the problems that cause a particular event or predict future result. Episode mining has been of great interest in many applications, including Internet anomaly intrusion detection [17, 22], biomedical data analysis [4, 23], stock trend prediction [24] and drought risk management in climatology data sets [10]. Besides, there are also studies on how to identify significant episodes from statistical model [2, 5].

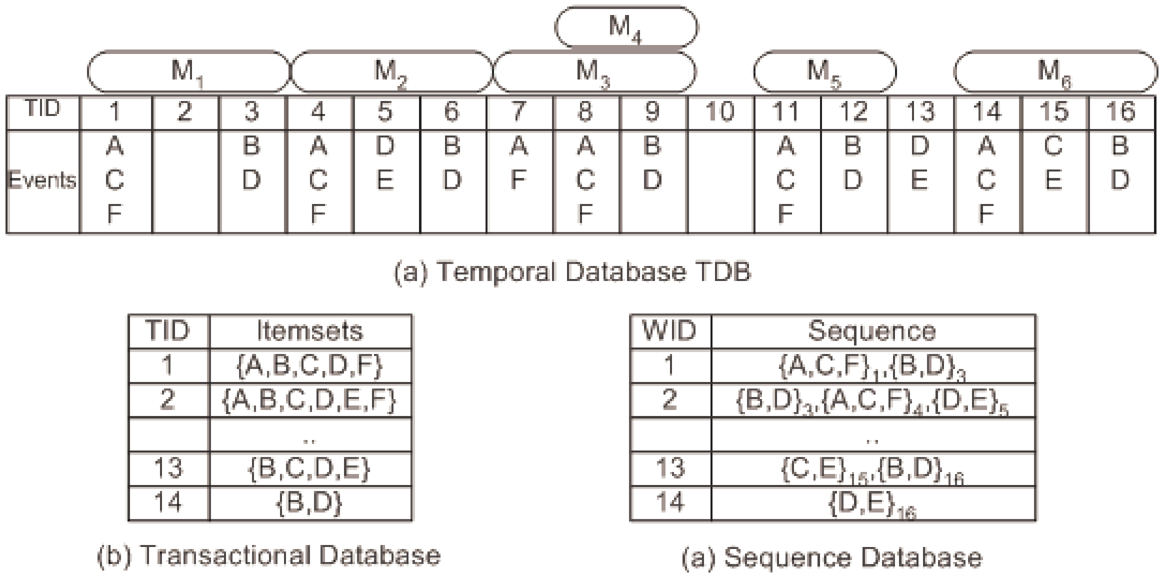
The task of mining frequent episodes was originally defined on "a sequence of events" where the events are sampled regularly as proposed by Mannila et al. [20]. Informally, an episode is a partially ordered collection of events occurring together. The user defines how close is close enough by giving the width of the time window win. The number of sliding windows with width win in a sequence is $t_e - t_s + win$, where t_s and t_e are called the starting interval and the ending interval, respectively. Take the sequence $S = A_3A_4B_5B_6$ (the subscript i represents the ith interval) as an example, there are 6-3+3=6 sliding windows in S, e.g. $W_1 = A_3$, $W_2 = A_3A_4$, $W_3 = A_3A_4B_5$, $W_4 = A_4B_5B_6$ and $W_5 = B_5B_6$, $W_6 = B_6$. Mannila et al. introduced three classes of episodes. *Serial episodes* consider patterns of a total order in the sequence, while *parallel episodes* have no constraints on the relative order of event sets. The third class contains composite episodes like serial combination of parallel episodes. In a way, serial and parallel episodes can be captured by sequential patterns and frequent itmesets respectively. Frequent itemsets to transaction databases are similar to parallel episodes, while sequential patterns to sequence databases are similar to serial episodes as defined in [20]. Therefore, we can mine parallel episodes by transforming an event sequence to a transaction database where each transaction are the unions of events from sliding window W_i. Similarly, we can mine serial episodes by transforming an event sequence to a sequence database, where each sequence are the serial combinations of events from W_i. However such methods are not efficient, since the space requirement is win times original database size. Finally, composite episodes can be mined from serial joins of parallel episodes.

Mannila et al. presented a framework for discovering frequent episodes through a level-wise algorithm, WINEPI [20], for finding parallel and serial episodes that are frequent enough. The algorithm was an Apriori-like algorithm based on the "anti-monotone" property of episodes' support. Unfortunately, this support count has a defect, i.e., the duplicate counting of an occurrence of an episode. For example, in the sequence $S = A_3A_4B_5B_6$, episode $< A >$ is supported by 4 sliding windows, while episode $< A, B >$ is matched by 2 sliding windows, e.g., $W_3 = A_3A_4B_5$, $W_4 = A_4B_5B_6$. Instead of counting the number of sliding windows that support an episode, Mannila et al. therefore consider the number of minimal occurrences of an episode from another perspective. They presented MINEPI [19], an alternative approach for the discovery of frequent episodes based on minimal occurrences (mo) of episodes. A minimal occurrence of an episode α is an interval such that no proper subwindow contains the episode α. For example, episode $< A >$ has mo support 2 (interval [3,3] and [4,4]) as the number of occurrences, while episode $< A, B >$ has only mo support 1 from interval [4,5]. However, both measures are not natural for the calculating of an episode rule's confidence (conditional probability). For example, the serial episode rule "When event A occurs, then event B occurs within 3 time units" should have probability or confidence 2/2 in the sequence $S = A_3A_4B_5B_6$ since every occurrence of A

is followed by B within 3 time units. Therefore, we need a measure that facilitates the calculating of such episode rules to replace the number of sliding windows or minimal occurrences. The problem has also been discussed in [16], but no algorithms are proposed.

In addition, we sometimes find several events occur (multi-variables) at one time slot in terms of various intervals (e.g., hours, days and weeks). We refer to such sequences as *complex sequences*. Note that a temporal database is also a kind of complex sequence with temporal attributes. Mining frequent episodes in a complex sequence has more extensive applications than in a simple sequence. Therefore, we discuss the problem on mining frequent episodes over complex sequence in this paper, where the support of an episode is modified carefully to count the exact occurrences of episodes. We propose two algorithms in mining frequent episodes in complex sequences, including MINEPI+ and EMMA. MINEPI+ is modified from previous vertical-based MINEPI [19] for mining episodes in a complex sequence. MINEPI+ employs depth first enumeration to generate the frequent episodes by *equalJoin* and *temporalJoin*. To further reduce the search space in pattern generation, we propose a brand new algorithm, EMMA (Episodes Mining using Memory Anchor), which utilize memory anchors to accelerate the mining task. Experimental evaluation on both real world and synthetic data sets shows that EMMA is more efficient than MINEPI+.

The rest of this paper is organized as follows. Section 2 reviews related work in sequence mining. We define the problem of frequent serial episode mining in Section 3. Section 4 presents our serial episode mining algorithms, including MINEPI+ and EMMA. (The extensions of the algorithms to parallel episode mining will be discussed in Appendix). Experiments on both synthetic and real world data


M$_1$
M$_2$
M$_3$
M$_4$
M$_5$
M$_6$

TID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Events	A C F		B D	A C F	D E	B D	A F	A C F	B D		A C F	B D	D E	A C F	C E	B D

(a) Temporal Database TDB

TID	Itemsets
1	{A,B,C,D,F}
2	{A,B,C,D,E,F}
	..
13	{B,C,D,E}
14	{B,D}

(b) Transactional Database

WID	Sequence
1	$\{A,C,F\}_1,\{B,D\}_3$
2	$\{B,D\}_3,\{A,C,F\}_4,\{D,E\}_5$
	..
13	$\{C,E\}_{15},\{B,D\}_{16}$
14	$\{D,E\}_{16}$

(a) Sequence Database

Figure 1. Temporal Database TDB

sets are reported in Section 5. Finally, conclusions are made in Section 6.

2. Related works

Mining significant patterns in sequence(s) is an important and fundamental issue in knowledge discovery area. For example, sequential patterns [1, 3, 7, 8, 26, 27, 30] consider the problem on discovering repeated subsequences in a database of sequences. On the other hand, some mining tasks focus on mining repeated subsequences in a sequence, e.g., frequent episodes [19, 20, 21], frequent continuities [14, 15, 28] and periodic patterns [6, 13, 18, 25, 29]. In this section, we distinguish various sequence mining tasks including sequential patterns, periodic patterns and frequent continuities which are related to frequent episodes. We also make an overall comparison between frequent itemsets and the four mining tasks.

The problem of mining sequential patterns was introduced in [1]. This problem is formulated as "Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user-specified $minsup$ threshold, sequential pattern mining is to find all frequent subsequences, whose occurrence frequency in the set of sequences is no less than $minsup$." The main difference between frequent itemsets and sequential patterns is that a sequential pattern considers the order between items, whereas frequent itemset does not specify the order. Srikant et al. proposed an Apriori-based algorithm, GSP (Generalized Sequential Pattern) [27] to the mining of sequential patterns. However, in situations with prolific frequent patterns, long patterns, or quite low $minsup$ thresholds, an Apriori-like algorithm may suffer from a huge number of candidate sets and multiple database scans. To overcome these drawbacks, Han et al. extend the concept of FP-tree [9] and proposed the PrefixSpan algorithm by prefix-projected pattern growth [26] for sequential pattern mining. In addition to algorithms based on horizontal formats, Zaki proposed a vertical-based algorithm SPADE [30]. SPADE utilizes combinatorial properties to decompose the original problem into smaller sub-problems that can be independently solved in main-memory using efficient lattice search techniques and simple join operations.

Periodic pattern, as suggested by its name, consider regularly appear events where the exact positions of events in the period are fixed [11, 12, 29]. To form periodicity, a list of k disjoint matches is required to form a contiguous subsequence where k satisfying some predefined minimum repetition threshold. For example, in Figure 1, pattern (A,*,B) is a periodic pattern that matches M_1, M_2, and M_3, three contiguous and disjoint matches, where event $\{A\}$ (resp. $\{B\}$) occurs at the first (resp. third) position of each match. The character "*" is a "don't care" character, which can match any single set of events. Note that M_6 is not part of the pattern because it is not located contiguously with the previous matches. To specify the occurrence, we use a 4-tuple (P, l, rep, pos) to denote a valid segment of pattern P

with period l starting from position pos for rep times. In this case, the segment can be represented by ((A,*,B), 3, 3, 1). Algorithms for mining periodic patterns also fall into two categories, horizontal-based algorithms, LSI [29], and vertical-based algorithms, SMCA [11, 12].

A continuity pattern is similar to a periodic pattern, but without the constraint on the contiguous and disjoint matches. For example, pattern [A,*,B] is a continuity with four matches M_1, M_2, M_3, and M_6 in Figure 1. The term continuity pattern was coined by Huang et. al. in [15] to replace the general term inter-transaction association defined by Tung, et al. in [28], since episodes and periodic patterns are also a kind of inter-transaction associations in the conceptual level. In comparison, frequent episodes are a loose kind of frequent continuities since they consider only the partial order between events, while periodic patterns are a strict kind of frequent continuities with constraints on the subsequent matches. In a word, frequent episodes are a general case of the frequent continuity, and periodic patterns are a special case of the frequent continuity. Two algorithms have been proposed for this task, including FITI and PROWL. FITI [28] is an Apriori-based algorithm which uses breadth-first enumeration for candidate generation and scans the horizontal-layout database. The PROWL algorithm [15], on the other hand, generates frequent continuities using depth first enumeration and relies on the use of both horizontal and vertical-layout databases.

Table 1 shows the comparison of the above mining tasks with frequent itemsets. The column “Order” represents whether the discovered pattern specify order; the column “Temporal” indicates whether the task is defined for a temporal database. According to the input database, frequent itemsets and sequential patterns are similar since they are defined on databases where the order among transactions/sequences

	Notation	Order	Temporal	Input	Constraint
Frequent Itemset	$I = \{i_1, \ldots, i_n\}$	N	N	a transaction DB	
Sequential Pattern	$S = I_1, \ldots, I_n$	Y	N	a sequence DB	
Serial Episode	$SEP =< I_1, \ldots, I_n >$	Y	Y	a sequence	
Parallel Episode	$PEP = \{I_1, \ldots, I_n\}$	N	Y	a sequence	
Frequent Continuity	$C = [I_1, \ldots, I_n]$	Y	Y	a sequence	[1]
Periodic Pattern	$P = (I_1, \ldots, I_n)$	Y	Y	a sequence	[1] [2]

[1] Fixed interval between I_i and I_{i+1}. [2] Contiguous match.

Table 1. Comparison of various pattern mining.

is not considered; whereas episodes, continuities, and periodic patterns are similar for they are defined on sequences of events that are usually sampled regularly. Frequent itemsets and sequential patterns are defined for a set of transactions and a set of sequences, respectively. Frequent itemsets show contemporal relationships, i.e., the associations among items within the same transaction; whereas sequential patterns present temporal/causal relationships among items within transactions of customer sequences. Finally, the differences of serial episodes, parallel episodes, periodic patterns, continuities are summarized in Table 1 as discussed above.

3. Problem definition

In this section, we first define the problem of frequent serial episode mining. We also discuss the parallel episode in section 4.3. Let E be a set of all events. An eventset is a nonempty subset of E. An input sequence can be represented as $(X_1, X_2, \ldots, X_O)$ where X_i is an eventset that occurs in i-th time interval or empty. The input sequence can also be described using a more general concept like a temporal database, where each tuple (t_j, X_{t_j}) records the time interval t_j for each event set X_{t_j} (nonempty). We refer this representation as horizontal format (e.g., Figure 1(a)). Let N be the number of tuples in the temporal database TDB. We say TDB has length N in O observation time intervals. We say that an event set Y is supported by a record (t_i, X_{t_i}) if and only if $Y \subseteq X_{t_i}$. An event set with k events is called a k-eventset.

Let $maxwin$ be the maximum window bound. When mining episode rules, only the rules which span less than or equal to $maxwin$ intervals will be mined. Users can thus use this mining parameter to avoid mining rules that span across too many intervals.

Definition 3.1 *A **sliding window** W_i in a temporal database TDB is a block of $maxwin$ continuous records along time interval, starting from interval t_i (where TDB contains an eventset at t_i-th time interval). Each interval t_{i_j} in W_i is called a subwindow of W_i denoted as $W_i[j]$, where $j = t_{i_j} - t_i$. Therefore, TDB with length N can be divided into N sliding windows, such as W_1=$(X_{t_1}, X_{t_1+1}, \ldots, X_{t_1+maxwin-1})$, W_2=$(X_{t_2}, X_{t_2+1}, \ldots, X_{t_2+maxwin-1})$, . . ., W_N=(X_{t_N}).*

Example 3.1 *Figure 1(a) shows a temporal database TDB with $fourteen$($N = 14$) transactions located at intervals 1, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15 and 16. Assume a value of 3 is set for maximum window $maxwin$. From definition, the number of sliding windows in Figure 1(c) is $fourteen$, from W_1, W_2, . . ., W_{14}. This will form a sequence database of size 14, which is different from the 18 sliding windows defined in [20] where empty intervals are considered. Thus, window W_1 has three subwindow $W_1[0]$ (containing events A, C and F), $W_1[1]$ (containing no event), and $W_1[2]$ (containing events B and D). As another example, window W_2 has also three subwindows $W_2[0] = X_3$, $W_2[1] = X_4$, and $W_2[2] = X_5$. Figure 1(b) shows the transaction database where each transaction is the union of the events in all subwindows of a window W_i, while Figure 1(c) shows the sequence database where each* *sequence represents a sliding window (with subscript denoting the time interval).*

Definition 3.2 *A **serial episode** is a nonempty partial ordered set of events $P =< p_1, p_2, \ldots, p_k >$ where each p_i is a nonempty eventset and p_i occurs before p_j for $i < j$. We call P is k-tuple serial episode or has length k.*

Definition 3.3 *Given two sequences $S =< s_1, s_2, \ldots, s_n >$ and $S' =< s'_1, s'_2, \ldots, s'_m >$, we say that S is a **super-sequence** of S' (i.e., S' is a **sub-sequence** of P) if and only if, each s'_j can be mapped by s_{i_j} ($s'_j \subseteq s_{i_j}$) and preserve it's order $(1 \leq i_1 \leq i_2 \leq \ldots \leq i_m \leq n)$*

For example, sequence $S_1 = < \{A, B\}, \{C\}, \{D, E\} >$ is a super-sequence of sequence $S_2 = < \{A\}, \{D\} >$, since the pattern $\{A\}$ ($\{D\}$, resp.) is a subset of $\{A, B\}$ ($\{D, E\}$, resp.). On the contrary, $S_3 = < \{A\}, \{C, D\} >$ is not a sub-sequence of S_1, since the pattern $\{C, D\}$ can not map to any itemset in S_1.

Definition 3.4 *Given a serial episode $P =< p_1, p_2, \ldots, p_k >$ and window bound w, we say a sliding window $W_i = (X_{t_i}, X_{t_i+1}, \ldots, X_{t_i+w-1})$ in TDB **supports** P if and only if, $p_1 \subseteq X_{t_i}$ and $< p_2, \ldots, p_k >$ is a subsequence of $(X_{t_i+1}, \ldots, X_{t_i+w-1})$. W_i is also called a **match** of the serial episode P. The number of sliding windows that match episode P is called the support count of P in temporal database TDB.*

Let us return to the previous example in Figure 1(a) and assume $maxwin = 3$, the serial episode $< A, D >$ is matched by sliding windows starting from time slot $1, 4, 7, 8, 11$ and 14. Therefore, there are six matches with respect to serial episode $< A, D >$. Note that the sequence corresponding to window $W_2 =< X_3, X_4, X_5 >$, although contains $< A, D >$, does not support this episode for the first subwindow does not contain A.

Definition 3.5 *The **concatenation** of two serial episodes $P =< p_1, \ldots, p_{l_1} >$ and $Q =< q_1, \ldots, q_{l_2} >$ is defined as $P \cdot Q =< p_1, \ldots, p_{l_1}, q_1, \ldots, q_{l_2} >$. P is called a **prefix** of $P \cdot Q$.*

Definition 3.6 *An **episode rule** generated from episodes is an implication of the form $X \Rightarrow Y$, where*

1. *X, Y are episodes with length l_1 and l_2, respectively.*
2. *The concatenation $X \cdot Y$ is an episode with length $l_1 + l_2$.*

Similar to the studies in mining typical association rules, episode rules are governed by two interestingness measures: support and confidence.

Definition 3.7 *Let N be the number of transactions in the temporal database TDB. Let $Match(X \cdot Y)$ be the number of support counts with respect to episode $X \cdot Y$ and $Match(X)$ be the number of support count with respect to episode X. Then, the **support** and **confidence** of an episode rule $X \Rightarrow Y$ are defined as*

$$Support = \frac{Matches(X \cdot Y)}{N}, \; Confidence = \frac{Matches(X \cdot Y)}{Matches(X)}. \tag{1}$$

If we only consider the episode that occurs at least $minsup$, there will have a problem. Take a simple sequence $S = A_1A_2B_3$ and $maxwin = 3$ as an example, we will find a 2-tuple episode $P =< A, B >$. It will be generated as a rule like "$< A > \Rightarrow < B >$". However, only A is a frequent item and B is not a frequent one. Therefore, we assume that not all frequent episodes has significance information, only the frequent episodes that all frequent itemsets in each tuples may be generated as a significant episode rule. To avoid generating the insignificant episodes. Therefore, the frequent episodes in this paper is defined as following.

Definition 3.8 *An episode $P =< p_1, \ldots, p_k >$ is a* ***frequent episode*** *if and only if the supports of P and all $p_i(1 \leq i \leq k)$ are at least the required user-specified minimum supports (i.e., $minsup$).*

Example 3.2 *Let the user-specified threshold minimum support $minsup$ and minimum confidence $minconf$ be 30 percent and 100 percent respectively. An example of a serial episode rule with maximum time window bound $maxwin = 3$ from the temporal database in Figure 1(a) will be:*

$$< ACF > \Rightarrow < BD > .$$

This rule "eventset $\{B, D\}$ occurs within two interval after eventset $\{A, C, F\}$" holds in the temporal database TDB with $support = 35.7\%(5/14)$ and $confidence = 100\%(5/5)$.

As in classical association rule mining, when frequent episodes and their support are known, the episode rule generation is straightforward. Hence, the problem of mining episode rules is reduced to the problem of determining frequent episodes and their support. Therefore, the problem is formulated as follows: given a minimum support level $minsup$ and a maximum window bound $maxwin$, our task is to mine all frequent episodes from temporal database with support greater than $minsup$ and window bound less than $maxwin$.

4. Mining Serial Episodes

In this section, we propose two algorithms for serial episode mining. We first show how to extend existing algorithm MINEPI to find the support counts instead of minimal occurrences in a complex sequence. Then, a new algorithm EMMA is proposed for more efficient mining of serial episodes from complex sequences. The comparison of the two algorithms is given in section 4.3.

4.1 MINEPI+

MINEPI is an iteration-based algorithm which adopts breadth first manner to enumerate longer serial episodes from shorter ones. But instead of scanning the temporal database (in horizontal format) for support counting, MINEPI compute the minimal occurrences mo of each candidate episode from the minimal occurrences of its subepisode by temporal joins. For example, we want to find all frequent serial episodes from a simple sequence $S = A_1A_2B_3A_4B_5$ with $maxwin = 4$ and $minsup = 2$. MINEPI first finds frequent 1-episode and records the respective minimal occurrence, i.e. $mo(A) = \{[1,1],[2,2],[4,4]\}$, $mo(B) = \{[3,3],[5,5]\}$ (We call this representation of the temporal sequence as vertical format). Using temporal join, we get intervals [1,3], [2,3], [2,5] and [4,5] for candidate 2-tuple episode $< A, B >$. Since $[1,3]$ and $[2,5]$ are not minimal, the minimal occurrences of $< A, B >$ will be $\{[2,3],[4,5]\}$.

Continuing the above example, if we want to count the number of sliding windows that match serial episode $< A, B >$, interval $[1,3]$ should be retained since the first subwindow contains A. Therefore, we have support count 3 for serial episode $< A, B >$ since [2,3] and [2,5] denote the same sliding window.

To extend MINEPI for our problem, we also need equal join for dealing with complex sequences. We will use these intervals to compute the right support count for the problem.

Definition 4.1 *Given the maximum window bound $maxwin$, the* ***bound list*** *of a serial episode P= $< p_1, \ldots, p_k >$, is the set of intervals $[ts_i, te_i]$ ($te_i - ts_i < maxwin$) such that $p_1 \subset X_{ts_i}$, $p_k \subset X_{te_i}$ and $[X_{ts_i+1}, X_{ts_i+2}, \ldots, X_{te_i-1}]$ is a super-sequence of $< p_2, \ldots, p_{k-1} >$. Each interval $[ts_i, te_i]$ is called a matching bound of P. By definition, the bound list of an event Y is the set of intervals $[t_i, t_i]$ such that X_{t_i} support Y.*

Given a serial episode $P =< p_1, \ldots, p_k >$ and a frequent 1-pattern f and their matching bound lists, e.g., $P.boundlist = \{[ts_1, te_1], \ldots, [ts_n, te_n]\}$ and $f.boundlist = \{[ts'_1, ts'_1], \ldots, [ts'_m, ts'_m]\}$. The operation **equal join** of P and f which computes the bound list for a new serial episode $P_1 =< p_1, \ldots, p_k \bigcup f >$ (denoted by $P \odot f$) is defined as the set of intervals $[ts_i, te_i]$ such that te_i = ts'_j for some j ($1 \leq j \leq m$). Similar to equal join, the operation **temporal join** (concatenation) of P and f (denoted by $P \cdot f$) which computes the bound list for new serial episode $P_2 =< p1, \ldots, p_k, f >$ is defined as the set of intervals $[ts_i, te'_j]$ such that $te'_j - ts_i < maxwin$, and $te'_j > te_i$ for some j ($1 \leq j \leq m$).

Lemma 4.1 *The support count of a serial episode P equals to the number of distinct starting positions of the bound list for P, denoted by EntityCount(P.boundlist).*

Example 4.1 *Let $maxwin = 4$, we use the matching bound lists $< A >$ .boundlist = {[1,1], [4,4], [7,7], [8,8], [11,11], [14,14]}, $< B >$ .boundlist = {[3,3], [6,6], [9,9], [12,12], [16,16]},and $< C >$ .boundlist = {[1,1], [4,4], [8,8], [11,11], [14,14], [15,15]} as an example. The matching bound*

```
Procedure of **MINEPI+(temporal database** $TD$, $minsup$, $maxwin$)
1. Scan $TD$ once, find frequent 1-episode $F_1$ and the $boundlists$;
2. for each $f_i$ in $F_1$ do
3.     SerialJoins($f_i$, $f_i.boundlist$, $f_i$);

Subprocedure of SerialJoins($\alpha$, $boundlist$, $lastItem$)
4. for each $f_j$ in $F_1$ do
5.     if ( $f_j > lastItem$ ) then
6.         $tempBoundlist = equalJoin(\alpha, f_j)$;
7.         if ($EntityCount(tempBoundlist) \geq minsup * |TDB|$) then
8.             SerialJoins($\alpha \odot f_j, tempBoundlist, f_j$);
9.     end if
10.    $tempBoundlist = temporalJoin(\alpha, f_j)$;
11.    if ($EntityCount(tempBoundlist) \geq minsup * |TDB|$) then
12.        SerialJoins($\alpha \cdot f_j, tempBoundlist, f_j$);
```

Figure 2. MINEPI+: Vertical-Based Frequent Serial Episode Mining Algorithm

list of equal join $(< A > \cdot < C >)$ *and temporal join* $(< B > \cdot < A >)$ *are* $< AC >$ *.boundlist =* $\{$*[1,1], [4,4], [8,8], [11,11], [14,14]*$\}$ *and* $< B, A >$ *.boundlist =* $\{$*[3,4], [6,7], [6,8], [9,11], [12,14]*$\}$*, respectively. The serial episode* $< B, A >$ *is matched by five matching bounds in four sliding windows, i.e., [3,6], [6,9], [9,12] and [12,15].*

Different from MINEPI, we apply depth first enumeration in pattern generation for memory saving. This is because breadth first enumeration must keep track of records for all episode in two consecutive levels, while depth first enumeration needs only to keep intermediate records for episodes generated along a single path. Figure 2 outlines the proposed MINEPI+ algorithm. The input to the procedure are a temporal database, minimum support threshold $minsup$ and maximum window bound $maxwin$. As the Definition 3.8, the frequent episode is generated by frequent itemsets. Therefore, before applying depth first enumeration, we scan temporal database TDB once, find frequent 1-episode F_1 and the boundlists (line 1). Frequent episodes are then generated by joining the boundlists of an existing episode (line 2–3) and an f_j in F_1 (line 4) through procedure call to $SerialJoins$. To avoid duplicate enumeration for equal joins, we define an order (e.g., alphanumerical order) in the events E. If the order of f_j is greater than the order of the $lastitem$ in the episode, we apply equal join (line 5–6) and check if the new serial episode $\alpha \cdot f_j$ is frequent or not (line 7). If the result is true, all frequent episodes which have prefix $\alpha \odot f_j$ (line 8) will be enumerated by recursive call to subprocedure $SerialJoins$. Similarly, we apply temporal join to the existing serial episode and the f_js in F_1 to get $\alpha \cdot f_j$ in line 10–12. We illustrate the MINEPI+ algorithm using the following example.

Example 4.2 *Given $minsup$ = 30% (5 times) and $maxwin$ = 4, the frequent 1-episodes F_1 for Figure 1(a) include $< A >$, $< B >$, $< C >$, $< D >$ and $< F >$. Due to space limitation, we only use episodes $< A >$ and $< C >$ as an example. The flowchart of executive process is shown in Figure 3. In the beginning, we check the equal and temporal join for $< A >$ and $< A >$. Since they have the same order, we only apply the temporal join for them. The entity count for $temporalJoin(< A >, < A >)$ = $\{$[1,4], [4,7], [7,8], [8,11], [11,14]$\}$ is 5 ($minsup * |TDB|$). Thus, we call subprocedure $SerialJoins$ for generating serial episodes which have prefix $< A, A >$. Next, we check serial episodes $< A, A, A >$, $< A, AC >$ and $< A, A, C >$. In this layer, all of them are infrequent. The recurse stops and backs to the prior procedure. Next, We compute the matching bound list for serial episode $< AC >$ by $equalJoin(< A >, < C >)$. The matching bounds of $< AC >$ are $\{$[1,1], [4,4], [8,8], [11,11], [14,14]$\}$. The recursive step then enumerates $< AC, A >$ and $< AC, C >$ (see Figure 3). Finally, only five serial episodes $< A >$, $< A, A >$, $< AC >$, and $< A, C >$ are outputted in this example.*

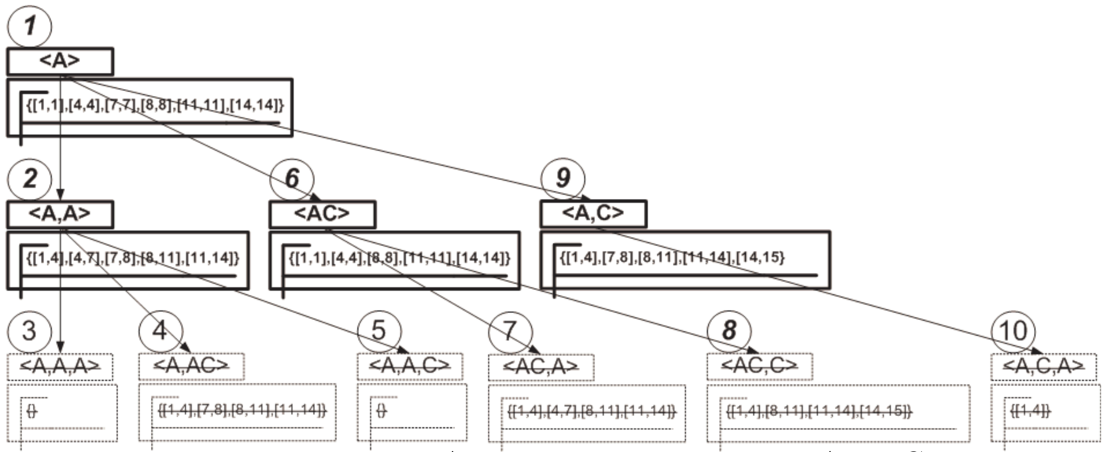


Figure 3. Flowchart for prefix $< A >$ (only demonstrate event A and C

Though the extension of MINEPI discover all frequent serial episodes, MINEPI+ has the following drawbacks:

- **A huge amount of combinations/computations**: Let $|I|$ be the number of frequent 1-episodes, WINEPI+ needs $|I|^2$ and $\frac{|I|^2-|I|}{2}$ checking for temporal joins and equal joins, respectively. For example, if there are 1000 frequent 1-episodes, the combinations is approximately 1.5 million times. Moreover, when the number of matching bounds increases, MINEPI+ requires more time in computations.

- **Unnecessary joins**: Since long episodes are generated from shorter ones, sometimes MINEPI+ makes some unnecessary checking. Take the bound list of serial episode $< A, A >$ in example 4.2 as an example. In this case, only the time bound $[7, 8]$ can be extended by temporal join to generate long episode since other bounds already reach the limits of maximum windows. Since the number of the extendable matching bounds for serial episode $< A, A >$ is less than $minsup * |TDB|$, we can skip all temporal joins for this prefix. We will discuss a pruning strategy in the following section.

- **Duplicate joins**: Furthermore, MINEPI+ also perform some duplicate checking. For example, to find serial episode $< ABCD, ABCD, ABCD >$, MINEPI+ needs 9 times of equal join (e.g., $equalJoin(< A >, < B >)$, $equalJoin(< AB >, < C >)$ and $equalJoin(< ABC >, < D >)$, etc.) and 2 temporal join (e.g., $temporalJoin(< ABCD >, < A >)$ and $temporalJoin(< ABCD, ABCD >, < A >)$). However, if we maintain the bound list for $< ABCD >$, we only needs 2 temporal join.

4.2 EMMA

In this section we propose an algorithm, EMMA (**E**pisode <u>M</u>ining using <u>M</u>emory <u>A</u>nchor), that overcomes the drawbacks of the MINEPI+ algorithm described in the previous section. According to the definition of frequent episodes in Definition 3.8, we have a idea for the sake of reducing in duplicate checking. Therefore, EMMA is divided into 3-phases, including

1. **Frequent itemset mining**: Mining frequent itemset in the complex sequence.
2. **Database encoding**: Encode each frequent itemset with a unique ID and construct them into a encoded horizontal database.
3. **Frequent serial episode mining**: Mining frequent serial episodes in the encoded database.

Similar to the algorithm FITI (First-Intra-Then-Inter) for frequent continuity mining [28], the idea is avoid duplicate checking by mining all frequent itemsets (Phase I) and use them to form frequent serial

Item	Timelist	Item	LocationList
$\{A\}$	$\{1, 4, 7, 8, 11, 14\}$	$\{A\}$	$\{0, 5, 11, 13, 18, 24\}$
$\{B\}$	$\{3, 6, 9, 12, 16\}$	$\{B\}$	$\{3, 9, 16, 21, 28\}$
$\{C\}$	$\{1, 4, 8, 11, 14, 15\}$	$\{C\}$	$\{1, 6, 14, 19, 25, 27\}$
$\{D\}$	$\{3, 5, 6, 9, 12, 13, 16\}$	$\{D\}$	$\{4, 8, 10, 17, 22, 23, 29\}$
$\{F\}$	$\{1, 4, 7, 8, 11, 14\}$	$\{F\}$	$\{2, 7, 12, 15, 20, 26\}$

(a) Two vertical formats using Tid (left) and Index (right)

Index	(Item, Tid)	Index	LocationList	Index	LocationList
0	(A, 1)	10	(D, 6)	20	(F, 11)
1	(C, 1)	11	(A, 7)	21	(B, 12)
2	(F, 1)	12	(F, 7)	22	(D, 12)
3	(B, 3)	13	(A, 8)	23	(D, 13)
4	(D, 3)	14	(C, 8)	24	(A, 14)
5	(A, 4)	15	(F, 8)	25	(C, 14)
6	(C, 4)	16	(B, 9)	26	(F, 14)
7	(F, 4)	17	(D, 9)	27	(D, 15)
8	(D, 5)	18	(A, 11)	28	(B, 16)
9	(B, 6)	19	(C, 11)	29	(D, 16)

(b) Horizontal format indexed by Tid

Figure 4. Indexed Database for Figure 1(a)

episodes. Thus, instead of frequent 1-itemsets, we have a larger set of all frequent itemsets as frequent 1-tuple episodes. Again, we will use the boundlists for each frequent 1-tuple episode to enumerate longer frequent episodes. However, we only combine existing episodes with a "local" frequent 1-tuple episode to overcome the huge amount of candidate generation. Now, in order to discover local frequent 1-tuple episode efficiently, we construct an encoded database indexed by time (Phase II) and utilize the boundlists as a memory anchor to access the time (or horizontal) based information. Finally, we use depth first enumeration to enumerate frequent serial episodes and carefully avoid unnecessary joins in Phase III.

```
Procedure of FIMA(temporal database $TDB$, $minsup$)
1. Scan $TDB$, find frequent 1-item $F_1$;
2. Scan $TDB$, transform $TDB$ into indexed database $IndexDB$
   and maintain the locations of all $F_1$ in the index database;
3. for each $f_i$ in $F_1$ do
4.     fimajoin($f_i, f_i.LocationList$);

Subprocedure of fimajoin($Pattern$, $LocationList$)
5. LFI = local frequent 1-item in $Pattern.PList$;
6. for each $lf_j$ in $LFI$ do
7.     fimajoin($Pattern \cup lf_j, lf_j.LocationList$);
```

Figure 5. FIMA: Frequent Itemset mining using Memory Anchor

4.2.1 Frequent itemset mining

There are already a lot of frequent itemset mining algorithms. Since the third phase of serial episode mining requires the time lists of each frequent itemset, we prefer using a vertical-based mining algorithm, e.g., Eclat [31]. However, similar to the drawbacks of MINEPI+, there are unnecessary candidate generation in the computation of Eclat. Therefore, we devise a more efficient algorithm FIMA (Frequent Itemset mining using Memory Anchor) which validates local frequent items to reduce the unnecessary combinations of existing frequent itemsets with nonlocal frequent items. To accelerate the validation of local frequent items, a horizontal format of the database which records the items indexed by time is necessary.

Here, we transform those transactions which contain frequent items to an array of 2-tuples, (I, Tid), sorted by the transaction ID Tid of the item I, where I is a frequent 1-item (see Figure 4(b)). Thus, instead of recording the timelist for each frequent item, we records the indexes of the frequent item in the array, as shown in Figure 4(a). Note that if there is only one frequent item in a transaction, such a tuple can be ignored to further save space. For example in Figure 1(a), item $\{A\}$, $\{C\}$ and $\{F\}$ are frequent 1-items at time slot 1, hence the transformation is needed. However, there is only one frequent item $\{D\}$ in time slot 5, thus the transformation is ignored.

The main frame of the FIMA is outlined in Figure 5. First of all, we scan database once and find frequent 1-items F_1 (line 1). Next, we transform the database into an array of 2-tuple $(Item, Tid)$ sorted by Tid and then $Item$. Then, we maintain the $LocationLists$ of each item in F_1 as searching anchors. For each f_i in F_1, we call subprocedure $fimajoin$ to extend longer itemsets with prefix f_i (line 3–4). In the subprocedure $fimajoin$, we find all local frequent 1-items lf_j by examining the transactions of current itemset (line 5). For example, if we want to extend A with locationlist $\{0, 5, 11, 13, 18, 24\}$, we will examine those tuples at $\{1, 2, 6, 7, 12, 14, 15, 19, 20, 25, 26\}$ since these tuples have the same Tid as A. The local frequent 1-items in this list (called projected list) are $\{C\}$ and $\{F\}$ with counts 5 and 6 respectively. Thus, new frequent itemset are generated by uniting A with one lf_j (line 6–7). Formally, the projected list of an location list is defined as follows.

Definition 4.2 *Given the location list of an itemset I, $I.LocationList$ = $\{t_1, t_2, \ldots, t_n\}$ in the index database $IndexDB$, the* **projected location list** *($PList$) of I is defined as $I.PList$ = $\{t'_1, t'_2, \ldots, t'_m\}$, where $t'_j.TID = t_i.TID$ for some t_i and $t_i < t_j \leq t_{|IndexTD|}$).*

The subprocedure $fimajoin$ is applied recursively to enumerate all frequent itemsets with known frequent itemsets as their prefixes. The recursive call stops when no more frequent itemsets are generated. With local frequent items, we reduce a lot of unnecessary joins of the existing frequent itemset with any

ID	Item	boundlist
#1	$\{A\}$	{[1,1],[4,4],[7,7],[8,8],[11,11],[14,14]}
#2	$\{B\}$	{[3,3],[6,6],[9,9],[12,12],[16,16]}
#3	$\{C\}$	{[1,1],[4,4],[8,8],[11,11],[14,14],[15,15]}
#4	$\{D\}$	{[3,3],[5,5],[6,6],[9,9],[12,12],[13,13],[16,16]}
#5	$\{F\}$	{[1,1],[4,4],[7,7],[8,8],[11,11],[14,14]}
#6	$\{A,C\}$	{[1,1],[4,4],[8,8],[11,11],[14,14]}
#7	$\{A,C,F\}$	{[1,1],[4,4],[8,8],[11,11],[14,14]}
#8	$\{A,F\}$	{[1,1],[4,4],[7,7],[8,8],[11,11],[14,14]}
#9	$\{B,D\}$	{[3,3],[6,6],[9,9],[12,12],[16,16]}

(a) Encoding table of the frequent itemsets for Figure 1(a)

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ID	#3 #7		#9	#3 #7		#9		#3 #7	#9		#3 #7	#9		#3 #7	#3	#9

(b) Encoded horizontal database for IDs #3, #7 and #9 in TDB

Figure 6. Location list and encoding table

frequent 1-items. The gain in time is a tradeoff of the cost in space as many algorithms. We will see such tradeoffs applied in serial episode mining in the following sections.

4.2.2 Encoded database construction

In the second phase, we associate each frequent itemset with a unique **ID** and construct a horizontal database EDB composed of these IDs. As shown in Figure 6(b)), EDB records the set of frequent itemsets (IDs) that occur at each time slot. To simplify the example, we only present the encoded database for IDs #3, #7 and #9 in TDB. Thus, we have for each frequent itemset the timelist from phase I and the encoded database constructed from phase II. Note that the timelists of the frequent itemsets are equivalent to the boundlists for frequent 1-tuple episodes. We show the boundlists for each frequent 1-tuple episode in Figure 6(a).

```
Procedure of EMMA(temporal database $TDB$, $minsup$, $maxwin$)
1. Call FIMA to find all frequent itemsets $FP_1$ and their timelists;
2. Associate each itemset with an #ID and construct the encoded database $EDB$ ;
3. for each $fid_i$ in frequent IDs $FP_1$ do
4.     emmajoin($fid_i, fid_i.boundlist$);

Procedure of emmajoin($Episode$, $boundlist$)
5. Find local frequent IDs $LFP$ in $Episode.PBL$ and their boundlists;
6. for each $lf_i$ in $LFP$ do
7.     Output $Episode \cdot lf_i$;
8.     if $(ExtCount(lf_i.boundlist) \geq minsup * |TDB|)$
9.         emmajoin($Episode \cdot lf_i, lf_i.boundlist$);
```

Figure 7. EMMA: Frequent Serial Episode Mining Using Memory Anchor

4.2.3 Frequent serial episode mining

The complete algorithm of EMMA is illustrated in Figure 7. Line 1 and 2 represent Phase I and II, respectively. Similar to MINEPI+, it adopts depth first enumeration to generate longer serial episodes (line 3-4, 6-7). However, EMMA generates only frequent serial episodes by joining an existing serial episodes with local frequent IDs (line 5). This is accomplished by examining the those transactions followings the matching bounds of current serial episode. For example, if we want to extend $\#3 = \{C\}$ with boundlist $\{$[1,1], [4,4], [8,8], [11,11], [14,14], [15,15]$\}$, we need to count the occurrences of IDs in the following bounds not exceeding $maxwin = 4$, i.e. [2,4], [5,7], [9,11], [12,14], [15,16] and [16,16]. When examining the IDs in these bounds, we also record the boundlists of IDs. Thus, when new serial episodes are generated by temporal joining (line 7), we know their boundlists immediately. Formally, the projected bound list of an location list is defined as follows.

Definition 4.3 *Given the bound list of a serial episode P, $P.boundlist$ = $\{[ts_1, te_1], \ldots, [ts_n, te_n], \}$ in the encoded database ED, the* **projected bound list** *(PBL) of P is defined as $P.PBL$ = $\{[ts'_1, te'_1], \ldots, [ts'_n, te'_n], \}$ where ts'_i=$min(ts_i + 1, |TDB|)$ and $te'_i = min(ts_i + maxwin - 1, |TDB|)$.*

Furthermore, if the number of extendable bounds for a serial episode P are less than $minsup*|TDB|$, then we can skip all extensions of the prefix P (line 8). For example, if the boundlist of a serial episode α is {[1,3], [3,5], [8,11], [11,14], [14,15]} and $maxwin = 4$, the extendable bounds include {[1,3], [3,5], [14,15]} since [8,11] and [11,14] already reach the maximum window bound. If the $minsup$ is 5, we don't need to extend serial episode α then. This strategy can avoid some unnecessary checking spent in MINEPI+. The procedure $emmajoin$ is called recursively until no more new serial episodes can be extended. The operation of phase III in EMMA can be best understood by an illustrative example as described below.

Example 4.3 *Given three frequent IDs #3, #7, #9 with their $boundlist$ and the encoded database in Figure 6. Let $minsup$ and $maxwin$ be 5 and 4, respectively. For each frequent 1-tuple episode, i.e. ID, we call $emmajoin$ to extend prefix $< ID >$. For ID $\#3$, the projected bound list is $\#3.PBL$ = {[2,4], [5,7], [9,11], [12,14], [15,16], [16,16]}. By examining the transaction at these bounds in Figure 6, the matching bounds for $< \#3, \#3 >$, $< \#3, \#7 >$ and $< \#3, \#9 >$ are calculated respectively as {[1,4], [8,11], [11,14], [14,15]}, {[1,4], [8,11], [11,14]} and {[1,3], [4,6], [8,9], [11,12], [14,16]}. Note that the number of sliding windows (i.e. distinct start positions) in the three lists are 4, 3, and 5, respectively. Therefore, the frequent serial episodes generated from prefix $\#3$ are $< \#3, \#9 >$, e.g.*

$< C, ACF >$. Since the number of extendable bounds for this episode is 5, serial episode $< \#3, \#9 >$ can be extended by recursive call to procedure $emmajoin$. However, we found no local frequent IDs in $< \#3, \#9 > .PBL$. Therefore, we backtrack to prefix $< \#7 >$ and discover the local frequent IDs in $< \#7 > .PBL$. The extensions of the episodes can be mined by applying the above process recursively to each episodes.

4.3 Discussion

Since EMMA generates longer patterns based on shorter ones, it does not generate any candidate patterns for checking. However, EMMA needs to maintain vertical-based boundlists for each frequent itemset and a horizontal-based encoded database. Therefore, the memory requirement for EMMA is greater than MINEPI+. When the number of frequent itemsets grows are large, it is unrealistic to maintain all patterns in the main memory. There are two alternative solutions. Firstly, we can maintain the boundlists of frequent itemsets in disk, then read them sequentially for episode extension. Theoretically, the disk-based EMMA can reduce half the memory requirement than original EMMA. Secondly, as the suggestion in [14], we can mine closed frequent itemsets in phase I and generate compressed frequent episode or devise new algorithms for mining closed frequent episodes.

5. Experiments

In this section, we report the performance study of the proposed algorithms on both synthetic data and real world data. All the experiments are performed on a 3.2GHz Pentium PC with 1 Gigabytes main memory, running Microsoft Windows XP. All the programs are written in Microsoft/Visual C++ 6.0.

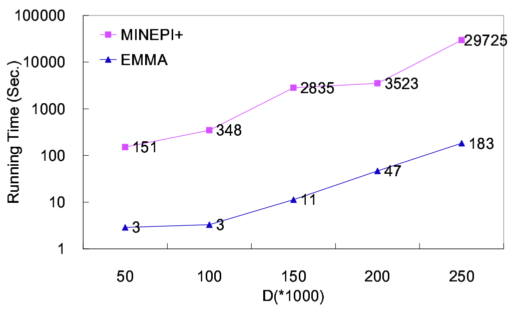
Sym	Definition	Default
$\|D\|$	# of time instants	100K
N	# of events	1000
T	Average transaction size	6
$\|C\|$	# of candidate continuities	2
L	Average continuity length	3
I	Average itemset length	3
W	Average window length	5
Sup	Average support	4%

Table 2. Meanings of symbols

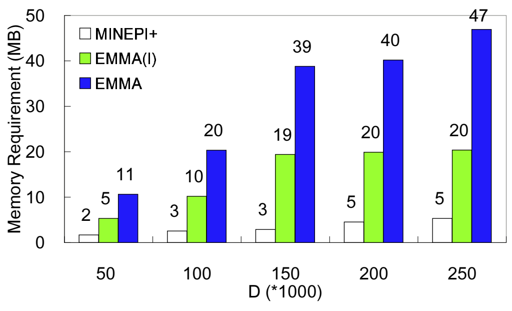
5.1 Synthetic data

For performance evaluation, we use synthetically generated temporal data, D, consisting of N distinct symbols and $|D|$ time instants. A set of candidate patterns C, is generated as follows. First, we decide the window length using geometrical distribution with mean W. Then L ($1 < L < W$) positions are chosen for non-empty event sets. The average number of frequent events for each time slot is set to I. The number of occurrences of a candidate continuity follows a geometrical distribution with mean $Sup * |D|$. A total of $|C|$ candidate patterns are generated. Next, we assign events to each time slot in D. The number of events in each time instant is picked from a Poisson distribution with mean T. For each time instant, if the number of the events in this time instant is less than T, the insufficient events are picked randomly from the symbol set N. Table 2 shows the notations used and their default values.

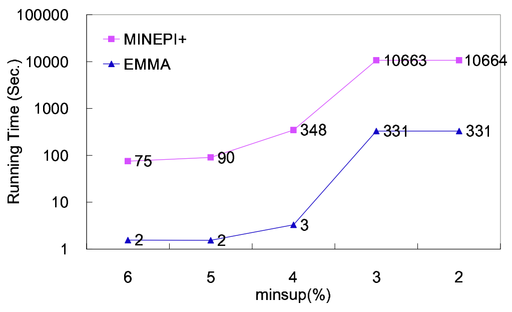
Figure 8 depicts the comparison results among MINEPI+ and EMMA for synthetic data with default parameter $minsup = 4\%$ and $maxwin = 5$. From Figure 8(a) we can see that when the data size increases, the gap between MINEPI+ and EMMA in the running time becomes more substantial. EMMA is faster than MINEPI+ (by a magnitude of 150 for $|D| = 250K$). However, EMMA requires more



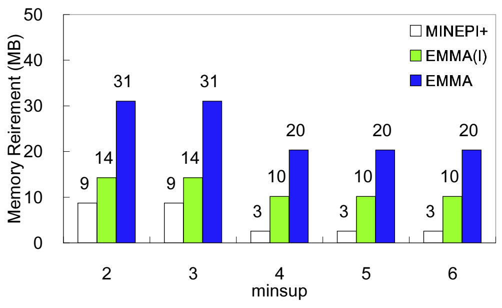
(a) Running Time v.s. Data Size



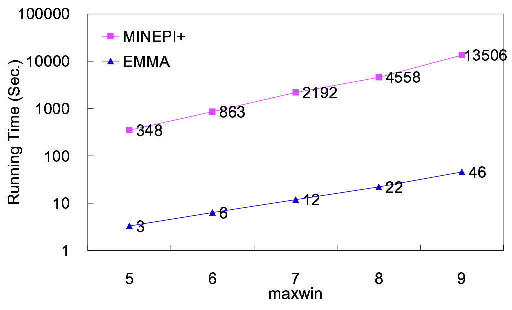
(b) Memory requirement v.s. Data Size



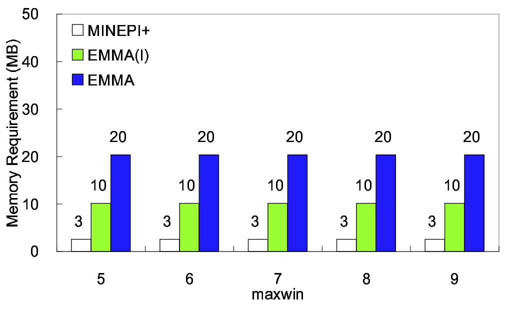
(c) Running Time v.s. *minsup*



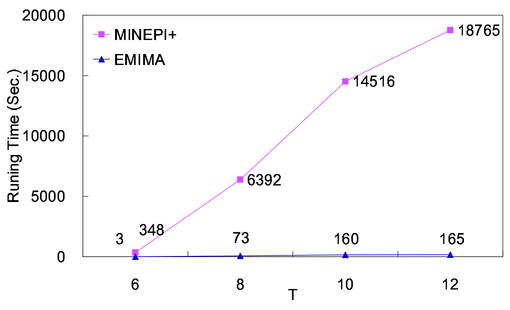
(d) Memory requirement v.s. *minsup*



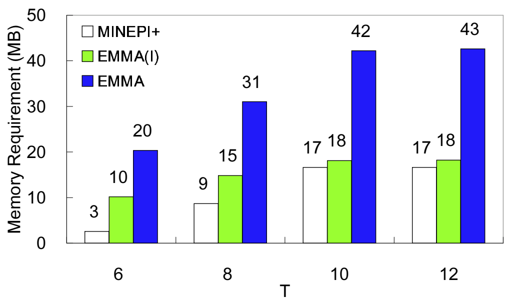
(e) Running Time v.s. *maxwin*



(f) Memory requirement v.s. *maxwin*



(g) Running Time v.s. T



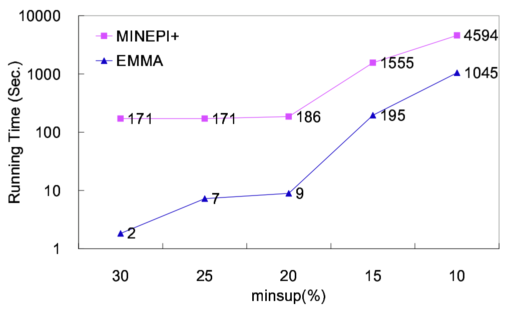
(h) Memory requirement v.s. T

Figure 8. Performance comparison in synthetic data

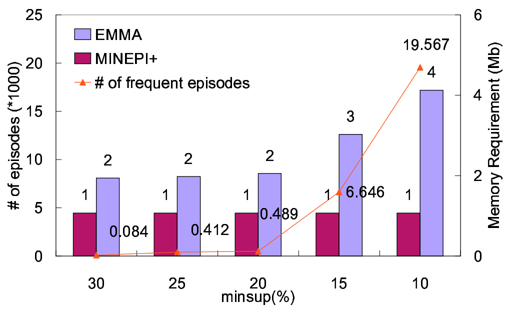
memory as shown in Figure 8(b). We also record the memory requirement of EMMA at phase I, denoted by EMMA(I). If the timelists of frequent itemsets are maintained in disk, the memory requirement will be $EMMA - EMMA(I)$. Therefore, EMMA(disk) needs approximate 27MB at $|D| = 250K$.

The runtime of MINEPI+ and EMMA on the default data set with varying minimum support threshold, $minsup$, from 2% to 6% is shown in Figures 8(c). Clearly, EMMA is faster and more scalable than MINEPI+, since the number of combinations in MINEPI+ grows rapidly as the $minsup$ decreases, while EMMA only considers the local frequent patterns in the projected bound lists. Again, the memory requirement for EMMA increases as $minsup$ decreases, since the number of frequent itemsets increases as $minsup$ decreases (see Figures 8(d)).

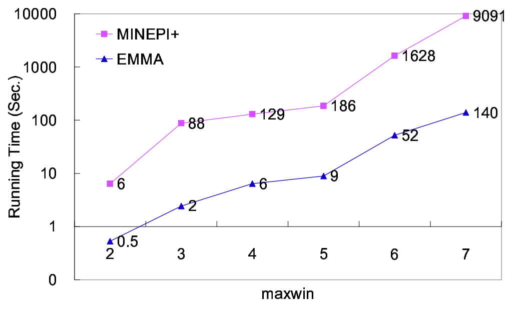
Figure 8(e) shows the scalability of the algorithms with varying maximum window. Both curves in Figure 8(e) go upwards because the number of frequent episodes increases exponentially as $maxwin$ increases. However, EMMA still outperforms MINEPI+ with varying $maxwin$. In Figures 8(f), the memory requirement is steady for both MINEPI+ and EMMA. Thus, the maximum window threshold does not affect the memory requirement a lot. In Figure 8(g), the total running time for MINEPI+ and EMMA are linear to the average transaction size T. However, for large transaction size, MINEPI+ requires significantly more time in equal join. In a word, the performance study shows that the EMMA algorithm is efficient and scalable for frequent episode mining, and is about an order of magnitude faster than MINEPI+. However, MINEPI+ requires smaller and stable memory space than EMMA.



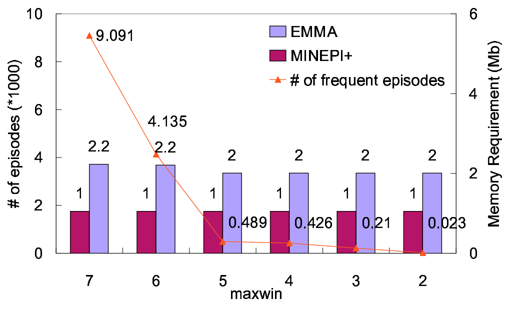
(a) Running Time v.s. $minsup$



(b) Memory Usage v.s. $minsup$



(c) Running Time v.s. $maxwin$



(d) Memory Usage v.s. $maxwin$

Figure 9. Performance comparison in real data

5.2 Real World Dataset

We also apply MINEPI+ and EMMA to a data set comprised of ten stocks (electronics industry) in the Taiwan Stock Exchange Daily Official list for 2618 trading days from September 5, 1994 to June 21, 2004. We discretize the stock price of go-up/go-down into five level: upward-high(UH): $>= 3.5\%$, upward-low(UL): $< 3.5\%$ and $> 0\%$, changeless(CL): 0%, downward-low(DL): $> -3.5\%$ and $< 0\%$, downward-high(DH): $<= -3.5\%$. In this case, the number of events in each time slot is 10, and the number of events is 50 (10*5). Figure 9(a) shows the running time with an increasing support threshold, $minsup$, from 10% to 30%. Figure 9(c) shows the same measures with varying $maxwin$. As the $maxwin$/$minsup$ threshold increases/decreases, the gap between MINEPI+ and EMMA in the running time becomes more substantial. Figures 9(b) and (d) show the memory requirements and the number of frequent episodes with varying $minsup$ and $maxwin$. As the $maxwin$ threshold increases or $minsup$ threshold decreases, the number of frequent episodes is increased. The memory requirement in MINEPI+ is steady. However, EMMA needs to maintain more frequent itemsets as the $minsup$ decreases; whereas the memory requirement with varying $maxwin$ in EMMA is changed slightly. MINEPI+ is better than EMMA in memory saving (by a magnitude of 4 for $minsup = 10\%$).

6. Conclusion and Future Work

In this paper, we discuss the problem of mining frequent episodes in a complex sequence and propose two algorithms to solve this problem. First, we modify previous vertical-based MINEPI [19] to MINEPI+ as the baseline for mining episodes in a complex sequence. To avoid the huge amount of combinations/compuataions and unnecessary/duplcate checking, we utilize memory to propose a brand-new memory-anchored algorithm, EMMA. (The extensions of the algorithms for parallel episode mining are given in Appendix.) The experiments show that EMMA is more efficient than MINEPI+ on both synthetic and real data set.

So far we have only discussed serial and parallel episodes. The combination of serial and parallel episodes remains to be solved. As suggested in [21], the recognition of an arbitrary episode can be reduced to the recognition of a hierarchical combination of serial and parallel episodes. However, there are some complications one has to take into account. Thus, further researches are required.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, pages 3–14, 1995.

[2] M. Atallah, R. Gwadera, and W. Szpankowski. Detection of significant sets of episodes in event sequences. In *Proceedings of Third IEEE International Conference on Data Mining (ICDM)*, 2004.

[3] M. N. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression of constraints. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(3):530–552, 2002.

[4] G.Casas-Garriga. Discovering unbounded episodes in sequential data. In *Proceedings of 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, 2003.

[5] R. Gwadera, M. Atallah, and W. Szpankowski. Reliable detection of episodes in event sequences. In *Proceedings of Third IEEE International Conference on Data Mining (ICDM)*, 2003.

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

[7] J. Han and J. Pei. Mining frequent patterns by pattern-growth: Methodology and implications. *ACM SIGKDD Explorations (Special Issue on Scalable Data Mining Algorithms)*, 2(2):14–20, 2000.

[8] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*, pages 355–359, 2000.

[9] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery: An International Journal (DMKD)*, 8(1):53–87, 2004.

[10] S. K. Harms, J. Deogun, J. Saquer, and T. Tadesse. Discovering representative episodal association rules from event sequences. In *Proceeding of 2001 IEEE International Conference on Data Mining. (ICDM'01)*, 2001.

[11] K. Y. Huang and C. H. Chang. Asynchronous periodic pattern mining in transactional databases. In *Proceedings of the IASTED International Conference on DATABASES AND APPLICATIONS (DBA'04)*, pages 17–19, 2004.

[12] K. Y. Huang and C. H. Chang. Mining periodic pattern in sequence data. In *Proceedings of 6th International Conference on Data Warehousing and Knowledge Discovery (DaWak'04)*, pages 401–410, 2004.

[13] K. Y. Huang and C. H. Chang. Smca: A general model for mining synchronous periodic pattern in temporal database. *IEEE Transaction on Knowledge and Data Engineering (TKDE)*, 2005. To Appear.

[14] K. Y. Huang, C. H. Chang, and K.-Z. Lin. Cocoa: An efficient algorithm for mining inter-transaction associations for temporal database. In *Proceedings of 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'04)*, volume 3202 of *Lecture Notes in Computer Science*, pages 509–511. Springer, 2004.

[15] K. Y. Huang, C. H. Chang, and K.-Z. Lin. Prowl: An efficient frequent continuity mining algorithm on event sequences. In *Proceedings of 6th International Conference on Data Warehousing and Knowledge Discovery (DaWak'04)*, volume 3181 of *Lecture Notes in Computer Science*, pages 351–360. Springer, 2004.

[16] K. Iwanuma, Y. Takano, and H. Nabeshima. On anti-monotone frequency measures for extracting sequential patterns from a single very-large data sequence. In *Proceedings of the First International Workshop on Knowledge Discovery in Data Streams, in conjunction with ECML/PKDD 2004*, 2004.

[17] J. Luo and S. M. Bridges. Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection. *International Journal of Intelligent Systems*, 15(8), 2000.

[18] S. Ma and J. Hellerstein. Mining partially periodic event patterns with unknown periods. In *Proceedings of the International Conference on Data Engineering (ICDE'01)*, pages 205–214, 2001.

[19] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 146–151, 1996.

[20] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 210–215, 1995.

[21] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in event sequences. *Data Mining and Knowledge Discovery (DMKD)*, 1(3):259–289, 1997.

[22] K. H. Min Qin. Frequent episode rules for internet anomaly detection. In *Proceedings of The Third IEEE International Symposium on Network Computing and Applications (NCA)*, 2004.
[23] N. L. N. Meger, C. Leschi and C. Rigotti. Mining episode rules in stulong dataset. In *ECML/PKDD 2004 Discovery Challenge (PKDD)*, 2004.
[24] A. Ng and A. W.-C. Fu. Mining frequent episodes for relating financial events and stock trends. In *Proceedings of The 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2003.
[25] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98)*, pages 412–421, 1998.
[26] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the International Conference on Data Engineering (ICDE'01)*, pages 215–226, 2001.
[27] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT'96)*, volume 1057 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 1996.
[28] A. K. H. Tung, H. Lu, J. Han, and L. Feng. Efficient mining of intertransaction association rules. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(1):43–56, 2003.
[29] J. Yang, W. Wang, and P. Yu. Mining asynchronous periodic patterns in time series data. *IEEE Transaction on Knowledge and Data Engineering (TKDE)*, 15(3):613–628, 2003.
[30] M. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.
[31] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 12(3):372–390, 2000.

Appendix

In this section, we define the problem of frequent parallel episodes mining and discuss how to modify our algorithms for this problem.

Definition 6.1 *A* ***parallel episode*** $I = \{i_1, \ldots, i_k\}$ *($i_j \in E$) is a set of events that occur within a window with length less than $maxwin$. We say parallel episode I is also a k-event parallel episodes.*

Definition 6.2 *Given a parallel episode $I = \{i_1, \ldots, i_k\}$ and the window bound win, we say the sliding window $W_i = (X_{t_i}, X_{t_i+1}, \ldots, X_{t_i+win-1})$ in TDB supports I if and only if, $I \subseteq U_i$ where*

$$U_i = \bigcup_{j=0}^{w-1} X_{t_i+j}$$

The number of sliding windows that match episode I is called the window count of I in the temporal database TDB.

Take parallel episode $I_1 = \{A, D\}$ in Figure 1(a) as an example, we find that I_1 is supported by twelve sliding windows (from W_1 to W_{12}). Thus, the parallel episode I_1 has 12 matches. Note that the number of windows that support an event can be as large as win times the number of occurrences for the event, since every event, except for those in the last $win - 1$ intervals, is counted by win windows. For example, E, although has only 3 appearances in Figure 1(a), is supported by 8 sliding windows. Thus, the minimum support for window counts should not be set too low, or there will be too many frequent 1-event parallel episodes.

Definition 6.3 *Given a $minsup$, we say an event x is* ***window frequent*** *if and only if it occurs at least $minsup$ sliding windows.*

The problem of frequent parallel episode mining is defined as discovering all parallel episodes that have at least $minsup$ support count within the maximum window bound win. Using vertical format representation, we shall maintain the sliding windows that support each window frequent parallel episodes (called **matching window lists**). Since we don't consider the order of events within a sliding window, we only needs to check the common parts of two known window lists when extending a short frequent episode.

First, we discuss the modified MINEPI+ for frequent parallel episode mining. Given a parallel episode $I = \{i_1, \ldots, i_k\}$, a window frequent 1-pattern wf and their matching window lists, e.g.,

```
Procedure of MINEPI2(temporal database $TDB$, $minsup$, $maxwin$)
1. Scan $TDB$ once, find window frequent items $WF_1$ and their matching $windowlists$;
2. for each $wf_i$ in $WF_1$ do
3.     ParallelJoins($wf_i$, $wf_i.windowlist$, $wf_i$);

Subprocedure of ParallelJoins($\alpha$, $windowlist$, $lastItem$)
4. for each $wf_j > lastItem$ in $WF_1$ do
5.     $tempWindowlist = windowJoin(\alpha, wf_j)$;
6.     if ($|tempWindowlist| \geq minsup * |TDB|$) then
7.         ParallelJoins($\alpha \bigcup wf_j, tempWindowlist, wf_j$);
```

Figure 10. MINEPI2: Vertical-Based Frequent Parallel Episode Mining Algorithm

$I.windowlist = \{IW_1, \ldots, IW_n\}$ and $wf.windowlist = \{FW_1, \ldots, FW_m\}$. The operation **windowJoin** of I and f which computes the window list for a new parallel episode $I' = \{i_1, \ldots, i_k, f\}$ (denoted by $I \bigcup f$) is defined as the intersection of the two window lists, $I.windowlist \bigcap wf.windowlist$. The modified MINEPI+ for parallel episodes is illustrated in Figure 10. To avoid the duplicate enumeration, we use alphabetical order to generate long parallel episodes (line 4). Starting from each window frequent event wf_i, all frequent parallel episodes with prefix wf_i can be enumerated by recursive calls to $ParallelJoins$.

Next, we discuss the modified solution of EMMA for frequent parallel episodes. The detailed modified algorithm, $EMMA2$, is illustrated in Figure 11. We shall see more clearly how EMMA differs from MINEPI in parallel episode mining. Instead of doing windowJoin directly in MINEPI, we will check local frequent items from the memory anchors, i.e. the window list of the current episode. To facilitate quick checking, we shall need more memory space as discussed in Section 4.3.

```
Procedure of **EMMA2(temporal database** $TDB$**,** $minsup$**,** $maxwin$**)**
1. **Find all window frequent items** $WF_1$ **and their** $windowlists$**;**
2. **for each** $wf_i$ **in** $WF_1$ **do**
4.      **WJoin(**$wf_i, wf_i.windowlist$**,**$wf_i$**);**

Procedure of **WJoin(**$\beta$**,** $windowlist$**,**$lastitem$**)**
5. **Find local window frequent items** $LF_1$ **in** $windowlist$ **and their** $windowlists$**;**
6. **for each** $lf_i > lastItem$ **in** $LF_1$ **do**
7.      **if** $(|lf_i.windowlist| \geq minsup * |TDB|)$
8.          **WJoin(**$\beta \bigcup lf_i, lf_i.windowlist$**,**$lf_i$**);**
```

Figure 11. EMMA2: Frequent Parallel Episode Mining Using Memory Anchor