JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 24, 1041-1064 (2008)

DOI:10.1688/JISE.2008.24.4.3

Efficient Discovery of Frequent Continuities by Projected Window List Technology*

KUO-YU HUANG, CHIA-HUI CHANG AND KUO-ZUI LIN
Department of Computer Science and Information Engineering
National Central University
Chungli, 320 Taiwan
E-mails: {want; kuozui}@db.csie.ncu.edu.tw; chia@csie.ncu.edu.tw

Mining frequent patterns is a fundamental problem in data mining research. A continuity is a kind of causal relationship which describes a definite temporal factor with exact position between the records. Since continuities break the boundaries of records, the number of potential patterns will increase drastically. An alternative approach is to mine compressed or closed frequent continuities (CFC). Mining CFCs has the same power as mining the complete set of frequent patterns, while substantially reducing redundant rules to be generated and increasing the effectiveness of mining. In this paper, we propose a method called projected window list (PWL) technology for the mining of frequent continuities. We present a series of frequent continuity mining algorithms, including PROWL+, COCOA and ClosedPROWL. Experimental evaluation shows that our algorithm is more efficient than previously works.

Keywords: data mining, frequent continuity, inter-transaction pattern, pattern growth, candidate-free enumeration

1. INTRODUCTION

Mining frequent patterns in databases is the fundamental for the data mining discipline. Recently, numerous studies have been made, including frequent itemsets [1, 2, 7], sequential patterns [3, 4, 6, 21], frequent episodes [16], periodic patterns [5, 8, 9, 17, 23], frequent continuities [10, 11, 22], causal relations [12, 20], *etc.* Some of the previous studies, such as those on frequent itemsets, are on mining contemporal relationships, *i.e.*, the associations among items within the same transaction where the transaction could be considered as the items bought by the same customer, events which happened on the same day, *etc.* For the sake of applications, measures such as confidence and correlation have been used to infer rules of the form "the existence of item A implies the existence of item B." For instance, a typical rule R_1 will be "if a customer buys butter, there is 80% confidence that he buys bread at the same time." The same concept can be applied to other applications as well, *e.g.* we can find rule R_2 in the stock market, such as "the prices of TSMC and UMC go up together on the same day with 80% probability." However, such rules indicate only statistical and contemporaneous relationships between items.

From the investors' point of view, a rule R_3 like "When the price of stock TSMC goes up for two consecutive days, the price of stock UMC will go up *on the third day* with 60% probability." may be more significant. This kind of causal rules between stocks

Received July 24, 2006; revised July 17, 2007; accepted September 5, 2007.
Communicated by Chung-Sheng Li.
* This work was sponsored by the National Science Council of Taiwan, R.O.C. under grant No. NSC 95-2524-S-008-001.

with definite temporal factors can be envisioned as a tool for describing and forecasting of the behavior of temporal databases. While a number of studies have been proposed for temporal relations, *e.g.* sequential patterns and frequent episodes, they have not considered definite temporal relationships. For instance, they can find a rule R_4 like "When the price of stock TSMC goes up, the price of stock UMC will go up *afterward*." The main difference between R_3 and R_4 is that R_3 describes the temporal factor clearly between events, whereas R_4 does not specify it.

The problem of mining association rules with definite temporal factor was defined by Lu *et al.* [13], using the term "*inter-transaction associations*" in contrast to intra-transaction associations for contemporary associations. The term was used because it breaks the boundaries of transactions to cross-record temporal associations. From this definition, mining frequent episodes [14, 15] and periodic patterns [8, 23] from sequences are kinds of inter-transaction as well, if the input sequences for the mining tasks are regarded as transactional databases. Thus, in order to distinguish the problem of Tung's from episodes and periodic patterns, we call such definite temporal associations "*continuity*" associations. A rule like R_3 can be generated from frequent continuities, an inter-transaction association which correlates the definite temporal relationships with each object.

Frequent continuities can be applied in several domains, including temporal, special databases, or other domains where the dimensional attribute is ordinal and can be divided into equal length intervals. For example, temporal intervals can be divided into days, months and seasons, *etc.* and special intervals can be divided into miles, regions and longitudes, *etc.* This type of pattern discovery can be used to study problem such as the evolution of stock prices and related populations.

Lu *et al.* [13] introduced inter-transaction association rules and developed two algorithms, E-Apriori and EH-Apriori, for mining inter-transaction associations by extension of Apriori. Besides, Tung *et al.* [22] proposed an algorithm, FITI (First Intra Then Inter), for the mining of frequent continuities. FITI is a 3-phase algorithm that uses the important property "A frequent continuity is composed of frequent intra-transaction itemsets and the don't-care(*)/mismatching characters." The first phase discovers intra-transaction itemsets. The second phase transforms the original database into another format to facilitate the mining of inter-transaction associations. The third phase follows the Apriori-based mining. In order to make search quickly, FITI is devised with several hashing structures for pattern searching and generation. Similar to Apriori-like algorithms, FITI could generate a huge number of candidates and require several scans over the whole database to check which candidates are frequent. Therefore, we [11] introduced a projected window list technique, PROWL, which enumerates new frequent continuities by checking frequent items in the following time slots of an existent frequent continuity. PROWL utilizes memory for storing both vertical and horizontal formats of the database, therefore it discovers frequent continuities without candidate generation. However, this algorithm was only applied to sequences of events instead of transactional databases.

Since continuities break the boundaries of records, the number of potential continuities and the number of rules will increase drastically. Therefore, we proposed the mining of compressed continuities [10] as an alternative idea. A compressed continuity is a continuity which is composed of only closed itemsets and the don't-care characters. In this paper, we summarize a series of algorithms using PROWL technique and advance one

step further to discover *closed* frequent continuities (CFC) which have no proper super-continuity with the same support in databases. Mining CFCs has the same power as mining the complete set of frequent continuities, while substantially reducing redundant rules to be generated and increasing the effectiveness of mining. The performance study shows that our algorithms are efficient and scalable for continuity mining, and are about an order of magnitude faster than the previous algorithm, FITI. The rest of this paper is organized as follows. Section 2 reviews related work in pattern mining. We define the problem of frequent continuities mining in section 3. Section 4 presents a series of our algorithms. Experiments on both synthetic and real world datasets are reported in section 5. Finally, conclusions are made in section 6.

2. RELATED WORKS

Over the past decade, many methods have been proposed to discover and explain aspects of behaviors hidden in temporal databases *TD* (see [19] for a survey). In this section, we distinguish four pattern mining tasks including sequential patterns, frequent episodes, periodic patterns and frequent continuities and make an overall comparison.

The problem of mining sequential patterns was introduced in [3]. 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 of the frequent subsequences, *i.e.*, the 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 [21]. However, an Apriori-like algorithm may suffer from handling a huge number of candidate sets and multiple database scans. To overcome these drawbacks, Han *et al.* extend the concept of Pattern-Growth and proposed the PrefixSpan algorithm by prefix-projected pattern growth [18] for sequential pattern mining.

The task of mining frequent episodes was defined on a sequence of event sets where the events are sampled regularly. An episode is defined to be a collection of events in a specific window interval that occur relatively close to each other in a given partial order [14]. Mannila *et al.* defined two kinds of episode: serial and parallel [14]. Serial episodes consider patterns with a specific order, while parallel episodes have no constraints on the relative order of eventsets. Take Fig. 1 for example and consider a window size of 3. There are seven matches of the serial episode $\langle AC, BD \rangle$, from E_1 to E_7, in the *TD* in Fig. 1. Meanwhile there are 13 matches of parallel episode $\{A, B, C, D\}$ which occurs in sliding window [1, 3], [2, 4], [3, 5], [4, 6], [5, 7], [6, 8], [7, 9], [8, 10], [9, 11], [10, 12], [11, 13], [12, 14] and [14, 16]. Mannila *et al.* also presented an Apriori-like algorithm, WINEPI [14], for finding all serial/parallel episodes that are frequent enough. They also presented MINEPI [15], an alternative approach for the discovery of frequent episodes based on minimal occurrences of episodes. Instead of counting the number of windows containing an episode, MINEPI looks at the exact occurrences of an episode and the relationships between those occurrences. Note that an episode considers only the partial order relation, instead of the actual positions, of events in a time window bound.

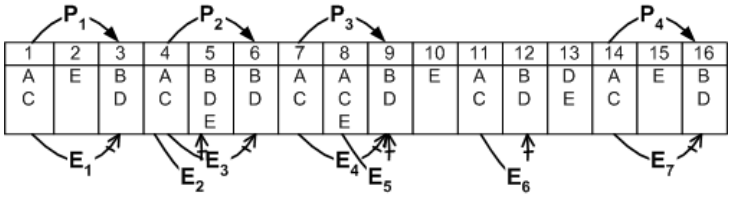


Fig. 1. An example of temporal database *TD*.

Unlike episodes, a periodic pattern considers not only the order of events but also the exact positions of events [8, 9, 23]. To form periodicity, a list of k disjoint matches is required to form a contiguous subsequence with k satisfying some predefined minimum repetition threshold. For example, in Fig. 1, pattern (AC, *, BD) is a periodic pattern that matches P_1, P_2, and P_3, three contiguous and disjoint matches, where eventset $\{A, C\}$ (resp. $\{B, D\}$) occurs at the first (resp. third) position of each match. Note that P_4 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 ((*AC*, *, *BD*), 3, 3, 1). Algorithms for mining periodic patterns also fall into two categories, horizontal-based algorithms, LSI [23], and vertical-based algorithms, SMCA [8, 9].

A continuity pattern is similar to a periodic pattern, but without the constraint on the contiguous and disjoint matches. For example, pattern [*AC*, *, *BD*] is a continuity with four matches P_1, P_2, P_3, and P_4 in Fig. 1. The term continuity pattern was coined by our pervious work in [11] to replace the general term inter-transaction association defined by Lu *et al*. in [13], since episodes and periodic patterns are also a kind of inter- transaction associations. 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. As noted in the introduction, two algorithms have been proposed for the task. FITI [22] is an Apriori-based algorithm which uses BFS enumeration for candidate generation and scans the horizontal-layout database. The PROWL algorithm [11], on the other hand, generates frequent continuities using DFS enumeration and relies on the use of both horizontal and vertical format.

Table 1. Comparison of various pattern mining capacities.

	Notation	Order	Temporal	Input	Constraint
Frequent Itemset	$\{i_1, \ldots, i_n\}$	N	N	a transaction DB	
Sequential Pattern	$I_1, \ldots, I_n$	Y	Y	a sequence DB	
Serial Episode	$\langle I_1, \ldots, I_n \rangle$	Y	Y	a sequence	
Parallel Episode	$\{I_1, \ldots, I_n\}$	N	Y	a sequence	
Frequent Continuity	$[I_1, \ldots, I_n]$	Y	Y	a sequence	[1]
Periodic Pattern	$(I_1, \ldots, I_n)$	Y	Y	a sequence	[1,2]

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

Table 1 shows the comparison of the above mining tasks with frequent itemsets. The "Order" represents whether the discovered pattern contains order; the "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 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 relationships among items within transactions of customer sequences. Finally, the differences of other patterns are summarized in Table 1 as discussed above.

3. PROBLEM DEFINITION

In this section, we define the problem of frequent continuity. We start from the definition of frequent continuity mining, then introduce the notion of compressed continuity and CFC, in turn. Let E be a set of all events. An event set is a non-empty subset of E. The input sequence can be described using a more general concept like a temporal database TD. A TD is a set of time records where each time record is a tuple (tid, X_i) for time instant tid and eventset X_i ($X_i \subseteq E$). Note that tid is an ordinal dimension and is divided into equal length interval. A sliding window W is a block of W continuous intervals along the time domain. A database stored in form of (tid, X_i) is called horizontal format (*e.g.* Fig. 1). We say that an event set Y is supported by a time record (tid, X_i) if and only if $Y \subseteq X_i$. An event set with k events is called a k-eventset.

Definition 1 A *continuity* with time window bound W is a nonempty sequence $P = [p_1, p_2, \ldots, p_W]$ where p_1 is an eventset and others are either an eventset or *, *i.e.* $p_j \subseteq E$ or {*} for $2 \le j \le W$.

The symbol "*" is introduced to allow mismatching (the "don't care" position in a pattern). Since a continuity can start anywhere in a sequence, we only need to consider patterns that start with a non-"*" symbol.

Definition 2 A *length* of a continuity $P = [p_1, p_2, \ldots, p_W]$ is the number of positions contains eventset in P.

A continuity P is called an L-continuity or has length L if exactly L positions in P contain eventset. For example, ["*AC*", *, *] is an 1-continuity; ["*AC*", *, "*BD*"] is a 2-continuity which has length 2.

Definition 3 Given a continuity $P = [p_1, p_2, \ldots, p_W]$ and a subsequence of W continuous slots $D = (d_1, d_2, \ldots, d_W)$ in TD (also called a sliding window), we say that D *supports* P if and only if, for each position j ($1 \le j \le l$), either p_j = * or $p_j \subseteq d_j$ is true. D is also called a match of P.

In general, given a *TD* and a pattern *P*, multiple matches of *P* may exist. In Fig. 1, P_1, P_2, …, P_4 are four matches of the continuity pattern [*AC*, *, *BD*].

Definition 4 The *concatenation* of two continuity patterns $P = [p_1, \ldots, p_{w_1}]$ and $Q = [q_1, \ldots, q_{w_2}]$ is defined as $P \cdot Q = [p_1, \ldots, p_{w_1}, q_1, \ldots, q_{w_2}]$. *P* is called a *prefix* of $P \cdot Q$.

Definition 5 An *inter-transaction association rule* generated from continuity patterns is an implication of the form $X \Rightarrow Y$, where

1. *X*, *Y* are continuities with window w_1 and w_2, respectively.
2. The concatenation $X \cdot Y$ is a continuity with window $w_1 + w_2$.

Similar to the studies in mining intra-transaction rules, continuity inter-transaction association rules are governed by two interestingness measures: support and confidence.

Definition 6 Let |*TD*| be the number of transactions in the *TD*. Let $Sup(X \cdot Y)$ be the number of matches with respect to continuity $X \cdot Y$ and $Sup(X)$ be the number of matches with respect to continuity *X*. Then, the *support* and *confidence* of an inter-transaction association rule $X \Rightarrow Y$ are defined as

$$Support = \frac{Sup(X \cdot Y)}{|TD|}, \quad Confidence = \frac{Sup(X \cdot Y)}{Sup(X)}. \tag{1}$$

Definition 7 A continuity *C* is a *frequent continuity* if and only if the number of supports of *C* is at least the required user-specified minimum supports (*i.e.*, *minsup*).

Example: Let threshold minimum support (*minsup*) and minimum confidence (*minconf*) be 25% and 60% respectively. An example on an inter- transaction association rule with maximum time window bound (*maxwin*) = 3 from the database in Fig. 1 will be: [*AC*, *] $\Rightarrow$ [*BD*]. This rule (Eventset {*B*, *D*} occurs two slots later after eventset {*A*, C}.) holds in the *TD* with *support* = 25% (4/16) and *confidence* = 67% (4/6).

As in classical association rule mining, if the frequent continuities and their support are known, the inter-transaction rule generation mining is straightforward. Hence, the problem of mining inter-transaction rules is reduced to the problem of determining frequent continuities and their support. Therefore, the problem is formulated as follows: given a *minsup* and a window bound *maxwin*, our task is to mine all frequent continuities from *TD* with support greater than *minsup* and window bound less than *maxwin*. Since frequent continuity mining often generates a very large number of frequent continuities, it hinders the effectiveness of mining. Therefore, we proposed an alternative idea to mine compressed/CFCs, which have the same power as mining the complete set of frequent continuities, while substantially reducing redundant pattern generation and increasing the effectiveness of the mining process.

Definition 8 A *compressed continuity* with time window bound *W* is a nonempty sequence $CP' = [cp_1, cp_2, \ldots, cp_W]$ where cp_1 is a closed frequent itemset and others are either closed frequent itemsets or *.

In Fig. 1, pattern [*AC*, *, *D*] is a compressed frequent continuity in Fig. 1, while pattern [*A*, *, *D*], [*C*, *, *D*] are not compressed frequent continuity patterns since event *A* and *C* are not closed frequent itemsets.

Definition 9 Given two continuities $P = [p_1, p_2, \ldots, p_u]$ and $P' = [p'_1, p'_2, \ldots, p'_v]$, we say that *P* is a *super-continuity* of *P′* (*i.e.*, *P′* is a *sub-continuity* of *P*) if and only if, for each non-* pattern p'_j ($1 \le j \le w$), $p'_j \subseteq p_{j+o}$ is true for some integer *o*. The integer *o* is also called the offset of *P* and $v + o \le u$.

For example, continuity *P* = [*AC*, *E*, *BD*] is a super-continuity of continuity *P′* = [*E*, *B*, *], since the pattern *E* (*B*, resp.) is a subset of *E* (*BD*, resp.) with offset 1. On the contrary, continuity *P″* = [*E*, *B*, *AC*] is not a sub-continuity of *P*, since *P″* can not map to *P* with a fixed offset. Note that if we don't consider the offset in the continuity matching, the continuity *P′* will not be a sub-continuity of continuity *P*.

Definition 10 A continuity $C = (c_1, c_2, \ldots, c_W)$ is a *CFC* if there exists no proper super-continuity of *C* that has the same support as *C* in database.

With CFCs, we can directly generate a reduced set of inter- transaction rules without having to determine all frequent continuities, thus reducing the computation cost.

4. THE ALGORITHMS

This section presents the algorithms for frequent/compressed/closed continuities.

4.1 PROWL+

In this section, we extend the PROWL [11] for frequent continuity mining in *TD*. Unlike Apriori-like algorithm, PROWL enumerates new frequent continuities by concatenating a frequent item in the projected window list of an existent frequent continuity using DFS enumeration. To facilitate this enumeration, PROWL utilizes memory for storing both the time slots for each event (called vertical formats, *e.g.* Fig. 2 (a)) and the events at each time slot (called horizontal formats, *e.g.* Fig. 1). To see how PROWL works, we defined the so called projected window list, starting from the definition of the timelist for continuity. Formally, the time list of a continuity pattern *P* records the sliding windows that support *P*, particularly, the last time slots of all matches.

Event	Time List
A	1, 4, 7, 8, 11, 14
B	3, 5, 6, 9, 12, 16
C	1, 4, 7, 8, 11, 14, 15
D	3, 5, 6, 9, 12, 13, 16
E	2, 5, 8, 10, 13, 15

(a) Vertical database layout.

ID	F.I.	Time List	Note
[1]	{*A*}	1, 4, 7, 8, 11, 14	
[2]	{*B*}	3, 5, 6, 9, 12, 16	
[3]	{*C*}	1, 4, 7, 8, 11, 14, 15	C.F.I.
[4]	{*D*}	3, 5, 6, 9, 12, 13, 16	C.F.I.
[5]	{*E*}	2, 5, 8, 10, 13, 15	C.F.I.
[6]	{*A*, *C*}	1, 4, 7, 8, 11, 14	C.F.I.
[7]	{*B*, *D*}	3, 5, 6, 9, 12, 16	C.F.I.

(b) Encoding table of the frequent itemsets.

Fig. 2. Vertical format and frequent itemsets for example database *TD*.

Definition 11 Given a *TD* and a continuity *P* with time window bound *W*, let I_i denote a subsequence of *W* time slots $I_i = (TD[t_i], TD[t_i + 1], \ldots, TD[t_i + W - 1])$ in *TD* that supports *P*. Assume there are *k* matches of *P* in *TD*. The *time list* of *P* is defined as $P.timelist = \{t_1 + W - 1, t_2 + W - 1, \ldots, t_k + W - 1\}$, *i.e.* the set of the last time slots of all matches.

By definition, each event is itself continuity with window size equal to one. The time list for an 1-continuity pattern is consistent with the time list for an event. Now, we define the projected window list of a pattern as follows.

Definition 12 Given the time list of a continuity *P*, $P.timelist = \{t_1, t_2, \ldots, t_k\}$ in *TD*, the *projected window list* (*PWL*) of *P* with offset *TD* is defined as $P.PWL_{td} = \{w_1, w_2, \ldots, w_k\}$, $w_i = t_i + td$ for $1 \le i \le k$. Note that a time slot w_i is removed from the projected list if w_i is greater than $|TD|$, *i.e.* $w_i \le |TD|$ for all *i*.

If an itemset *X* is frequent in the projected window list of pattern *P* with offset 1, we refer to the concatenation $P \cdot X$ as an *extension* of *P* and $P \cdot X.timelist = P.PWL_1 \cap X.timelist$. We call *X* a *frequent follower* of *P*. Take Fig. 1 as an example. The time list of continuity [*C*] is {1, 4, 7, 8, 11, 14, 15}, the projected window list is $P_{[C]}.PWL_1$ = {2, 5, 8, 9, 12, 15, 16}, which is also the time list for continuity [*C*, *] (the don't care character has a time list which includes all time slots). Therefore, the projected window list of [*C*, *] is $P_{[C,*]}.PWL_1$ = {3, 6, 9, 10, 13, 16}. Since [*D*] is a frequent item in $P_{[C,*]}.PWL_1$ (*min-sup* = 25%), we can concatenate pattern [*C*, *] with pattern [*D*] as a frequent continuity [*C*, *, *D*], which has the time list = {3, 6, 9, 13, 16}. In this way, PROWL discovers frequent continuities without candidate generation.

However, PROWL was designed only for sequences of events, not for eventset sequences where multiple events can occur at a time slot. The reason for this is that the pattern growing method considers only one single events instead of eventsets at one time. To extend PROWL to general *TD*, we utilize the important property that a frequent inter-transaction continuity pattern must be made up of frequent intra-transaction itemsets [22]. Therefore, The PROWL+ algorithm consists of three phases, including intra-transaction itemset mining, database transformation and inter-transaction continuity mining.

- The first phase of PROWL+ involves the mining of frequent intra-transaction itemsets. Since the third phase of the algorithm requires the time lists of each intra-transaction itemset, this phase is mined using a vertical mining algorithm, Charm [24], for frequent itemsets mining.
- The second phase is database transformation, where it encodes each frequent itemset (abbreviated F.I.) with a unique *ID* and constructs a recovered horizontal database composed of the IDs. To illustrate, Fig. 2 (b) shows the encoding table of F.I. in Fig. 1. Next, based on the time lists of the frequent itemsets together with the encoding table, we construct a recovered horizontal database as shown in Fig. 3.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Code	[1] [3] [6]	[5]	[2] [4] [7]	[1] [3] [6]	[2] [4] [5] [7]	[2] [4] [7]	[1] [3] [6]	[1] [3] [5] [6]	[2] [4] [7]	[5]	[1] [3] [6]	[2] [4] [7]	[4] [5]	[1] [3] [6]	[3] [5]	[2] [4] [7]

Fig. 3. Recovered horizontal database of *TD*.

```
Given the recovered horizontal database RD, the encoding table VD, minsup, maxwin;
Procedure of PROWL+()
1.  for each ID ID_i ∈ VD do
2.      Pattern[0] = ID_i;
3.      for j = 1 to maxwin − 1 do
4.          Pattern[j] = *;
5.      Project(VD[ID_i], Pattern, 1);
6.  end

Subprocedure of Project(TimeList, Pattern, Layer)
 1.  begin if (Layer ≤ maxwin) then
 2.      TimpVD.clear();
 3.      for each time instant T_i ∈ TimeList do
 4.          if (T_i < |RD|) then
 5.              PWL = T_i + 1;
 6.              for each ID ID_j ∈ RD[PWL] do
 7.                  TempVD[ID_j].insert(PWL);
 8.      begin for each ID ID_i ∈ TempVD do
 9.          if (TempVD[ID_j].size ≥ minsup) then
10.              Pattern[Layer] = ID_i;
11.              Project(TempVD[ID_i], Pattern, Layer + 1);
12.              Output Pattern;
13.          Pattern[Layer] = *;
14.          Project(TimeList, Pattern, Layer + 1);
15.      end
16.  end
```

Fig. 4. PROWL+: frequent continuity mining algorithm.

- In the third phase, we discover all frequent continuities from the recovered horizontal database by concatenating a frequent continuity with the F.I.s in its projected window lists using depth-first enumeration. Fig. 4 outlines the proposed PROWL+ algorithm. For each frequent 1-continuity P, or equivalently frequent itemset (ID), we calculate its projected window list with offset 1 from *P.timelist* and examine the time slots of *P.PWL* in *RD* (steps 3-7 in the *Project*) to find all followers, *i.e.* IDs. New frequent continuities are formed by concatenating P with a frequent follower (steps 9-10 of *Project*). The procedure *Project* is applied recursively to enumerate all continuities with known frequent continuities as their prefixes. The recursive call stops when the layer is greater than *maxwin* (step 1 of procedure *Project*).

We illustrate the PROWL+ algorithm using the following example.

Example: Given *minsup* = 25% (4 times) and *maxwin* = 3, the frequent itemsets for Fig. 1 include {*A*}, {*B*}, {*C*}, {*D*}, {*E*}, {*A*, *C*} and {*B*, *D*}. Each frequent itemset is encoded with a unique ID as shown in Fig. 2 (b). Then, we construct a recovered horizontal database composed of the IDs by the time lists of the frequent itemsets (see Fig. 3).

For ID [1], the projected window list is $P_{[1]}.PWL$ = {2, 5, 8, 9, 12, 15}, which is also the time list of continuity [[1], *]. By examining the time slots of $P_{[1]}.PWL$ in Fig. 3, the number of occurrences for [1], [2], [3], [4], [5], [6] and [7] in $P_{[1]}.PWL_1$ are calculated respectively as 1, 3, 2, 3, 4, 1 and 3. Since only [5] has sufficient support, others are sim-

ply ignored. Therefore, the frequent continuity generated from prefix [1] is [[1], [5]], *i.e.* [*A*, *E*], with 4 matches. Note that the frequent continuities should be decoded from its original symbol sets while it is being output. Using depth-first enumeration, we examine the frequent IDs in $P_{[[1],[5]]}.PWL$ = {3, 6, 9, 16} to extend the continuity [[1], [5]], where we acquire three continuities including [[1], [5], [2]], [[1], [5], [4]] and [[1], [5], [7]], each with 4 occurrences. Note that the projected window list of these three continuities is {4, 7, 10} (time record 17 (16 + 1) is greater than sequence length 16), they can not be extended to any longer, so the calls to procedure *Project* stop.

Recursively, we apply the above process to continuity [[1], *]. The projected window list of [[1], *] is $P_{[[1],*]}.PWL$ = {3, 6, 9, 10, 13, 16}. In this layer, we find the frequent IDs [2], [4] and [7] in time records of $P_{[[1],*]}.PWL$. Thus, three continuities are generated: [[1], *, [2]], [[1], *, [4]], and [[1], *, [7]], (*i.e.* [*A*, *, *B*], [*A*, *, *D*], [*A*, *, *BD*]). The extensions of the continuities can be mined by applying the above process respectively to each continuity. In summary, all frequent continuities with window 2, and having prefix [1] can be generated by concatenating [1] with a frequent event in $P_{[1]}.PWL$ or the don't care symbol. Similarly, we can find all frequent continuities having prefix [2] by constructing $P_{[2]}.PWL$ and mining them respectively. The set of frequent continuities is the collection of patterns found in the above recursive mining process.

4.2 COCOA

Since inter-transaction associations break the boundaries of transactions, the number of potential itemsets and the number of rules will increase drastically. This reduces not only efficiency but also effectiveness since users have to sift through a large number of mined rules to find useful ones. Thus, we apply closed frequent itemset mining instead of frequent itemset mining in the first phase to reduce the number of IDs [10]. For example, frequent itemsets {*A*} and {B} in Fig. 2 (b) can be ignored since they are not closed frequent itemsets, thus enumeration beginning with [1] and [2] can be eliminated. Therefore, we have a similar three-phase algorithm, COCOA, (Compressed Continuity Analysis), where the other two phases in COCOA are exactly the same as in PROWL+, while the first phase is replaced by CHARM [25] for closed frequent itemset mining. The result of the mining is the compressed continuities, as defined in Definition 8. Similar to COCOA, we can also replace the first phase of FITI by mining closed frequent itemset to discover compressed continuities. We call the corresponding algorithm ComFITI. The algorithm performances between COCOA and of ComFITI are compared in section 5.

4.3 ClosedPROWL

Although compressed continuity reduces the number of continuities, they are not the minimum set which represent all continuities. The ideal set is the CFC which is the target of ClosedPROWL. Similar to PROWL+ and COCOA, the ClosedPROWL algorithm also consists of three phases. In the first phase, we mine frequent closed itemsets (abbreviated C.F.I.). The second phase is the same as PROWL+, but only the closed patterns are transformed. The third phase of ClosedPROWL is outlined in Fig. 5. We apply two pruning strategies: **sub-itemset pruning** and **sub-continuity pruning** to reduce redundant enumeration in lines 9-11 and 15 of the ClosedProject procedure, respectively. The sub-itemset pruning strategy can be stated as follows.

```
Given recovered the horizontal database RD, Vertical database VD, minsup and maxwin;
Procedure of ClosedPROWL()
1.  for each ID ID_i ∈ VD do
2.      if (VD[ID_i].size ≥ minsup) then
3.          Pattern[0] = ID_i;
4.          for j = 1 to maxwin − 1 do
5.              Pattern[j] = *;
6.          Project(VD[ID_i], Pattern, 1);
7.  end

Subprocedure of ClosedProject(TimeList, Pattern, Layer)
 1.  begin if (Layer ≤ maxwin) then
 2.      TimpVD.clear();
 3.      PHTab.clear();      // Hash Table for SubItemset Pruning
 4.      for each time instant T_i ∈ TimeList do
 5.          if (T_i < |RD|) then
 6.              PWL = T_i + 1;
 7.              for each ID ID_j ∈ RD[PWL] do
 8.                  TempVD[ID_j].insert(PWL);
 9.      for each ID ID_i ∈ TempVD do
10.          if (TempVD[ID_j].size ≥ minsup) then
11.              SubItemsetPruning(TempVD[ID_i], PHTab);
12.      for each entity H_i ∈ PHTab do
13.          Pattern[Layer] = HI_i.ID;
14.          Project(TempVD[ID_i], Pattern, Layer + 1);
15.          SubContinuityPruning(H_i.Sup, Pattern, Layer + 1);
16.          Pattern[Layer] = *;
17.          Project(TimeList, Pattern, Layer + 1);
18.  end
```

Fig. 5. ClosedPROWL: closed frequent continuity mining algorithm.

Sub-itemset pruning: For two C.F.I. x and y in the project window list of a continuity P, if $Sup(P \cdot [x]) = Sup(P \cdot [y])$, the sub-itemset pruning works as following properties:

1. If $x \subset y$, then remove x since all extensions of $P \cdot [x]$ must not be closed.
2. If $x \supset y$, then remove y since all extensions of $P \cdot [y]$ must not be closed.
3. If $x.timelist = y.timelist$ and neither $x \subset y$ nor $x \supset y$, then remove both x and y, since all extensions of $P \cdot [x]$ and $P \cdot [y]$ must not be closed.

The correctness of the pruning can be proven by the following lemma and theorems.

Lemma 1 Let $P = [p_1, p_2, \ldots, p_w]$ and $Q = [q_1, q_2, \ldots, q_w]$ be two frequent continuities and $P.timelist = Q.timelist$. For any frequent continuity U, if $P \cdot U$ is frequent, then $Q \cdot U$ is also frequent, vice versa.

For example, the time lists of continuities $[C, D]$ and $[C, BD]$ in Fig. 1 are both $\{5, 9, 12, 16\}$. The probable frequent followers in the projected window list of pattern $[C, D]$ and $[C, BD]$ with offset 1 are the same, *e.g.*, $\{B\} : 1$, $\{D\} : 2$, $\{B, D\} : 1$, $\{E\} : 2$, $\{D, E\} : 1$ (the number after ":" indicate the support counts).

Theorem 1 Let $P = [p_1, p_2, \ldots, p_w, p_{w+1}]$ and $Q = [p_1, p_2, \ldots, p_w, p'_{w+1}]$ be two continuities. If $p_{w+1} \subset p'_{w+1}$ and $Sup(P) = Sup(Q)$, then all extensions of P must not be closed.

Proof: Since p_{w+1} is a subset of p'_{w+1}, wherever p'_{w+1} occurs, p_{w+1} occurs. Therefore, $P.timelist \supseteq Q.timelist$. Since $Sup(P) = Sup(Q)$, the equal sign holds, *i.e.* $P.timelist = Q.timelist$. For any extension $P \cdot U$ of P, there exists $Q \cdot U$ (Lemma 1), such that $Q \cdot U$ is a super-continuity of $P \cdot U$, and $(P \cdot U).timelist = P.PWL_{|U|} \cap U.timelist = Q.PWL_{|U|} \cap U.timelist = (Q \cdot U).timelist$. Therefore, $P \cdot U$ is not a CFC.

Theorem 2 Let $P = [p_1, p_2, \ldots, p_w, p_{w+1}]$ and $Q = [p_1, p_2, \ldots, p_w, p'_{w+1}]$ be two continuities. If $P.timelist = Q.timelist$ and neither $p_{w+1} \subset p'_{w+1}$ nor $p_{w+1} \supset p'_{w+1}$, then all extensions of P and Q must not be closed.

Proof: Consider the continuity $U = [p_1, p_2, \ldots, p_w, p_{w+1} \cup p'_{w+1}]$. $U.timelist = P.timelist \cap Q.timelist$. Since $P.timelist = Q.timelist$, we have $U.timelist = P.timelist = Q.timelist$. Using Theorem 1, all extensions of P and Q can not be closed because $Sup(U) = Sup(P) = Sup(Q)$.

In order to make the pruning efficient, we devise a hash structure, PHTab (prune header table) with *PHsize* buckets. All C.F.I.s with the same support counts are hashed into the same bucket. Each entry in the same bucket records a frequent ID x of the current continuity P, the time list of $P \cdot [x]$, and the support count of $P \cdot [x]$. The comparison of two frequent C.F.I. x and y in the projected window lists of a continuity P is restricted to the frequent IDs in the same buckets with the same support. As shown in Fig. 6, each new generated C.F.I. $Pattern(H_i.ID)$ must be examined by sub-itemset pruning strategy. Firstly, we initialize the bucket number *bkNum* as $H_i \rightarrow size\%PHSize$ (see line 1 in procedure SubItemsetPruning). If the C.F.I. of $H_i.ID$, $Pattern(H_i.ID)$, is the subset of some C.F.I. $Pattern(H_j.ID)$ in $PHTab[bkNum]$, property 1 is applied (lines 5-7). Conversely, we employ property 2 to delete the unnecessary C.F.I. in $PHTab[bkNum]$ if the new generated continuity is the superset of the C.F.I. (lines 8-10). To apply property 3, we remove both $H_i.ID$ and $H_j.ID$ from $PHTab[bkNum]$ if their timelists are equivalent (lines 11-14).The sub-itemset pruning technique removes the non-closed sub-continuity of CFC with zero offset since the pruning is invoked within a local search of a continuity. For those sub-continuities of CFCs with non-zero offset, they can still be generated in the mining process. Therefore, we need a checking step to remove non-CFCs. Again, a hash structure, FCTab, is devised to facilitate efficient sub-continuity checking as the following:

$$bkNum = Sup(P)\%BucketSize. \tag{2}$$

The procedure SubContinuityPruning is shown in Fig. 7. If the new generated continuity X has the same support with a sub-continuity Y in *FCTab*, Y can be pruned since Y is not a closed continuity (lines 5-7). On the other hand, if X is a sub-continuity of an existing one Y in *FCTab*, we simply ignore X (lines 8-10). We will use the following example to illustrate the mining process of ClosedPROWL and its corresponding flowchart is depicted in Fig. 8.

```
SubProcedure of SubItemsetPruning(TempVD[ID_i], PHTab)
 1. bkNum = TempVd[ID_i].size%BucketSize;
 2. IsClosed = ture;
 3. for each entry H_j ∈ PHTab[bkNum] do
 4.     if (TempVD[ID_i].size == H_j.sup) then
 5.         if (Pattern(ID_i) ⊂ Pattern(H_j.sup)) then
 6.             IsClosed = false;     // Prune Subtree of ID_i
 7.             break;
 8.         else if (Pattern(ID_i) ⊃ Pattern(H_j.ID)) then
 9.             Delete H_j;     // Prune Subtree of H_j.ID
10.             break;
11.         else if (TempVD[ID_i].timelist == TempVD[H_j.ID].timelist) then
12.             Delete H_j;     // Prune Subtree of ID_i and H_j
13.             IsClosed = false;
14.             break;
15.     if (IsClosed) then
16.         Add ID_i into PHTob[bkNum];
```

Fig. 6. SubItemsetPruning: sub-itemset pruning strategy.

```
SubProcedure of SubContinuityPruning(Sup, Pattern, Layer)
 1. bkNum = Sup%BucketSize;
 2. IsClosed = ture;
 3. for each entry P_i ∈ FCTab[bkNum] do
 4.     if (sup == P_i.sup) then
 5.         if (Pattern ⊂ P_i.Pattern) then
 6.             Delete P_i;     // Prune Subtree of H_j.ID
 7.             break;
 8.         else if (Pattern ⊃ P_i.Pattern) then
 9.             IsClosed = false;
10.             break;
11.     if (IsClosed) then
12.         Add Pattern into FCTab[bkNum];
```

Fig. 7. SubContinuityPruning: sub-continuity pruning strategy.

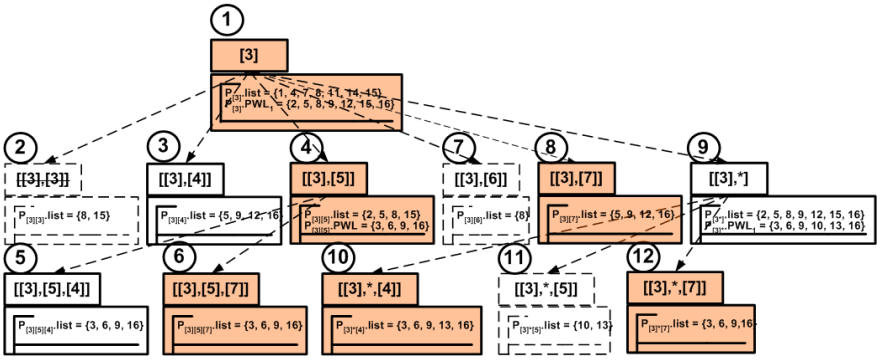


Fig. 8. Process of ClosedPROWL for prefix [3].

Example: Given *minsup* = 25% (4 times) and *maxwin* = 3, we discover all closed frequent continuities as follows. Let the bucket size of both hash structures be 4. Phase I and Phase II of ClosedPROWL produces closed frequent itemsets and recovered horizontal database, including the IDs [3], [4], [5], [6] and [7] (see Fig. 2 (b). The mining process for prefix [3] is shown in Fig. 8. By examining the time slots of $P_{[3]}.PWL_1$ the frequent continuities generated from prefix [3] are [[3], [4]], [[3], [5]] and [[3], [7]] with 4 matches. To apply the subitemset pruning, we insert these three continuities into the PHTab of prefix [[3]] at bucket *matches*%4. Since the ID [4] ({*D*}) is a sub-itemset of the ID [7] ({*BD*}), the continuity [[3], [4]] is removed (see Fig. 9 (a)). Therefore, only the continuities [[3], [5]] and [[3], [7]] are inserted into *FCTab*.

Bucket	Frequent ID	Sup	Time list
0	[4]	4	{5, 9, 12, 16}
	[5]	4	{2, 5, 8, 15}
	[7]	4	{5, 9, 12, 16}
1			
2			
3			

(a) PHTab of the prefix [*C*].

Bucket	Frequent ID	Sup	Time list
0	[4]	4	{3, 6, 9, 16}
	[7]	4	{3, 6, 9, 16}
1			
2			
3			

(b) PHTab of the prefix [*C*, *E*].

Fig. 9. Pruning header table (PHTab) for sub-itemset pruning.

Bucket	Continuity	Sup
0	[*C*, *E*]	4
	[*C*, *BD*]	4
	[*C*, *E*, *BD*]	4
	[*C*, *, *BD*]	4
1	[*C*, *, *D*]	5
2		
3	[*C*]	7

Fig. 10. Frequent continuity table (FCTab) for sub-continuity checking.

Next, the projected window list of {[3], [5]} is $P_{\{[3],[5]\}}.PWL_1$ = {3, 6, 9, 16}. In this layer, there exists two frequent IDs in $P_{\{[3],[5]\}}.PWL_1$, [4] and [7]. Again, we apply the sub-itemsets pruning to remove ID [4] (see Fig. 9 (b)) because [4] ⊂ [7] ({*D*} ⊂ {*B*, *D*}). For prefix [3], there are six potential closed continuities that insert into *FCTab*, including [3], [[3], [5]], [[3], [7]], [[3], [5], [7]], [[3], *, [4]], [[3], *, [7]] as shadow parts in Fig. 8. Finally, we apply the sub-continuity pruning to remove non-closed continuities, the frequent continuity table after mining in prefix [3] is shown as Fig. 10. Taking bucket 0 in *FCTab* as an example, since continuities [*C*, *E*] ([[3], [5]]) is a sub-continuity of the continuity [*C*, *E*, *BD*] ([[3], [5], [7]]), it can be removed. Similarly, [*C*, *, *BD*] is also a sub-continuities of [*C*, *E*, *BD*], so it can be removed. After employing the sub-continuity pruning, we find four potential closed frequent continuities of the prefix [*C*] ([3]), including [*C*], [*C*, *, *D*], [*C*, *E*] and [*C*, *E*, *BD*]. Note that these closed continuities are not all closed continuities in the overall data since they could be removed by their super-

continuities with the same support in other prefix pattern. For example, after mining process of prefix [4], [5], [6] and [7], we will find five closed frequent continuities, including [*C*], [*AC*, *, *BD*], [*AC*, *, *D*], [*AC*, *E*, *BD*] and [*C*, *BD*] of the *TD* in Fig. 1.

4.4 Correctness

We prove the correctness of the ClosedPROWL algorithm in this section. Since the search space of the PROWL and COCOA are the same as ClosedPROWL except for the pruning strategies, they can be proved similarly.

Lemma 2 The time list of a continuity $P = [p_1, p_2, \ldots, p_w]$ is $P.timelist = \bigcap_{i=1}^{w} p_i.PWL_{w-i}$.

We define the closure of an itemset p, denoted $c(p)$, as the smallest closed set that contains p. If p is closed, then $c(p) = p$. By definition, $Sup(p) = Sup(c(p))$ and $p.timelist = c(p).timelist$.

Theorem 3 A CFC is composed of only closed itemsets and don't care characters.

Proof: Assume $P = [p_1, p_2, \ldots, p_W]$ is a closed continuity, and some of the p_is are composed of non-closed itemsets. Consider the continuity $CP = [c(p_1), c(p_2), \ldots, c(p_W)]$, $CP.timelist = \bigcap_{i=1}^{w} c(p_i).PWL_{w-i} = \bigcap_{i=1}^{w} p_i.PWL_{w-i} = P.timelist$ (Lemma 2). Therefore, P is not a closed continuity. We thus have a contradiction to the original assumption that P is a closed continuity and thus conclude that "all closed continuities $P = [p_1, p_2, \ldots, p_W]$ are composed of only closed itemsets and the don't-care characters".

Theorem 3 is an important property as it provides a different view of mining closed frequent continuities. The observation tells us that instead of mining frequent itemsets, we can mine closed frequent itemsets before mining closed frequent continuities.

Theorem 4 The ClosedPROWL algorithm generates all closed frequent continuities.

Proof: First of all, the anti-monotone property "if a continuity is not frequent, all its super-continuities must be infrequent" is sustained for closed frequent continuities. According to Theorem 3, the search space composed of only closed frequent itemset covers all closed frequent continuities. ClosedPROWL's search is based on a complete set enumeration space. The only branches that are pruned as those that do not have sufficient support. The sub-itemset pruning only removed non-closed continuities (Theorem 1). Therefore, ClosedPROWL correctly identifies all closed frequent continuities. On the other hand, sub-continuity checking remove non-closed frequent continuities. Therefore, the ClosedPROWL algorithm generates all and only closed frequent continuities.

4.5 Extra-large

The proposed algorithms are basically memory-based algorithms, and their efficiency comes from the removal of database scans and candidate generation that are required by FITI. If the data is too large to fit in the memory space, a partition-and-validation strategy can be used to handle such a case. Suppose the *TD* is composed of *n* time

records, we divides the n time records into k partitions. Since the frequent continuities consider the cross-transaction patterns, each partition should has $maxwin - 1$ overlapping area to avoid losing patterns in generation. Each partition can be handled in memory by our algorithms. The local minimum support count for a partition is minsup multiplied by the number of time records in that partition, while the global minimum support count in a database is *minsup* multiplied by the number of time records in the database. Therefore, not all local frequent continuities are global frequent continuities. Such false-positive continuity patterns (frequent under local minimum support, but not frequent under global minimum support) must under go an additional scan in order to determine their support count. Take Fig. 11 as an example. Let *minsup* = 25% (4 times) and *maxwin* = 3, suppose the memory only maintain 7 time records each time. Therefore, we divides the 16 time record windows into 3 partitions, partitions 1 to 3, as shown in Fig. 11. Firstly, we mine the local frequent continuities which satisfied the local minimum support 25% ($\lceil 7 \times 25\% \rceil = 2$ times) in each partition. Finally, we scan the *TD* once to check which continuity is true-positive frequent continuities, satisfying the global minimum support of 25% ($\lceil 16 \times 25\% \rceil = 4$ times).

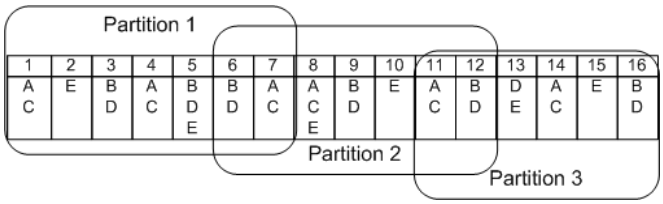


Fig. 11. Partition example.

4.6 Space Requirements and Improvement

Since the projected window list technique employs depth first enumeration for mining frequent continuities, it only generates longer patterns based on shorter ones. Specifically, it does not generate/maintain any candidate patterns for checking. However, ClosedPROWL needs to maintain generated continuities for subitemset and subcontinuity pruning. Compared with COCOA, we require additional memory to store *PHTab* and *FCTab*. Assume that average matches of each continuity and the number of potential continuities are t and n respectively. The maximum space requirement of *PHTab* is $n \times t$ (n entities in *PHTab*). Similarly, the space requirement in *FCTab* is $m \times t$, where m is the number of frequent closed continuities. When the number of closed continuities grows very large, it is unrealistic to maintain patterns in the main memory. To reduce the memory cost, we can apply only subitemset pruning and store the *FCTab* in disk, then read disk-resident buckets and apply sub-continuity pruning to remove non-closed continuities. The related experiments are demonstrated in the session below.

5. EXPERIMENTS

In this section, we report the performance study of the proposed algorithms on both synthetic data and real world data. Since the three phases of the proposed algorithms

have good correspond once with three phases of the FITI algorithm, it is possible to mine various continuities by combining various Phase Is with Phase III of FITI (FITI-3) and PROWL. The combinations are shown in Table 2. We already know the mining process of FITI and PROWL+, where the first phase involves frequent itemset mining. If we mine closed frequent itemsets at Phase I and apply FITI-3 or PROWL, we will get compressed frequent continuities. We call the algorithms ComFITI and COCOA, respectively. Finally, the closed frequent itemset mining at Phase I combined with PROWL and the pruning strategies at Phase III results the mining of ClosedPROWL for frequent closed continuities. We compare the five algorithms using synthetic data. All the experiments are performed on a 2.8GHz Pentium PC with 2.5 Gigabytes main memory, running Microsoft Windows/NT. All the programs are written in Microsoft/Visual C++ 6.0. In the following experiments, the size of PHTab and FCTab is set to 1000.

Table 2. Comparison of various mining tasks.

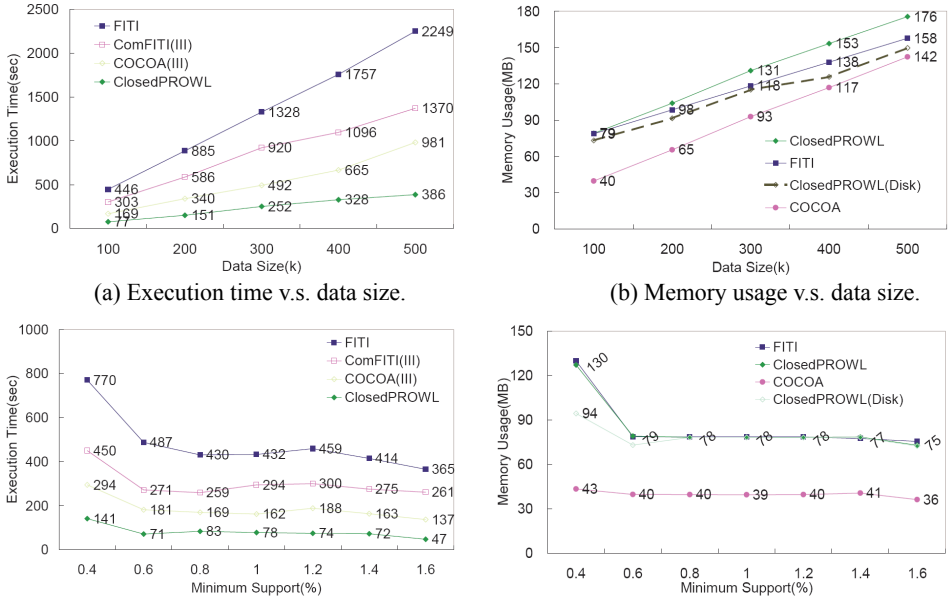
Mining Task	Phase I	Phase III	Algorithm
Continuity	Frequent Itemset	FITI-3	FITI
		PROWL	PROWL+
Compressed	Closed Frequent Itemset	FITI-3	ComFITI
		PROWL	CCOA
Closed		PROWL + Pruning	ClosedPROWL

Table 3. Meanings of symbols.

Sym	Definition	Default
$\|D\|$	# of time instants	100K
N	# of events	500
T	Average transaction size	10
$\|C\|$	# of candidate continuities	2
L	Average continuity length	3
I	Average itemset length	2
W	Average window length	5
Sup	Average support	2%

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 continuities 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 $Sup \times |D|$. A total of $|C|$ candidate continuities are generated. With all candidate continuities generated, we then 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 3 shows the notations used and their default values.



(a) Execution time v.s. data size. (b) Memory usage v.s. data size.

(c) Execution time v.s. *minsup*. (d) Memory usage v.s. *minsup*.

Fig. 12. Performance comparison I.

We start by looking at the performance of ClosedPROWL with default parameter *minsup* = 0.6% and *maxwin* = 5. Fig. 12 (a) shows the scalability of the algorithms with varying database size. ClosedPROWL is faster than FITI (by a magnitude of 5 for $|D|$ = 500K). The scaling with database size was linear. Therefore, the scalability of the projected window lists technique is proved. Another remarkable result is that COCOA performs better than ComFITI for the same mining task (compressed frequent continuity mining). The reason for the considerable execution time of FITI and ComFITI is that they must count the supports of all candidate continuities.

The memory requirement of the algorithms with varying database size is shown in Fig. 12 (b). In this case, the number of frequent continuities and closed frequent continuities are 13867 and 1183 respectively. The compression rate (# of closed frequent continuities /# of frequent continuities) is about 9%. As the data size increases, the memory requirement of ClosedPROWL, COCOA and FITI increases as well. However the memory usages of FITI and ClosedPROWL are about the same at $|D|$ = 100K and the difference is only 18MB at $|D|$ = 500K. As data size increases, ClosedPROWL requires more additional storage to maintain frequent continuities (*FCTab*). Therefore, we modify the algorithm as described in section 4.6 to disk-resident ClosedPROWL (labelled ClosedPROWL(Disk)). As illustrated in Fig. 12 (b), the memory requirement of the ClosedPROWL(Disk) is thus less than FITI but more than COCOA for subitemset pruning (*PHTab*). Since the number of closed frequent continuities is less (approximately 1000), the time cost between ClosedPROWL and ClosedPROWL(Disk) is very small (between 1-3 sec). Therefore, we didn't report the time cost of ClosedPROWL(Disk) Fig. 12.

The runtime and memory usage of FITI and ClosedPROWL on the default data set with varying minimum support threshold, *minsup*, from 0.4% to 1.6% are shown in Figs.

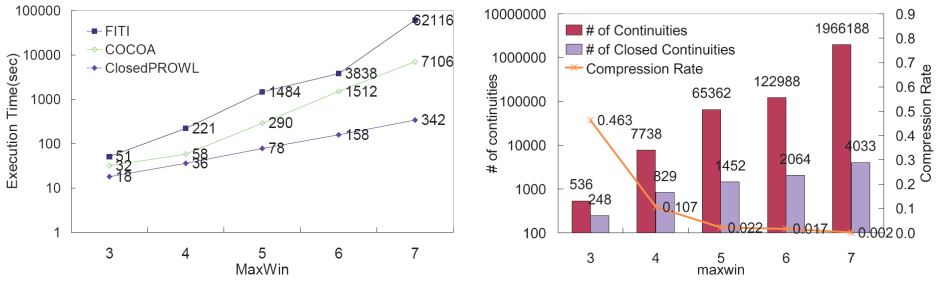
12 (c) and (d). Clearly, ClosedPROWL is faster and more scalable than both FITI and ComFITI with the same memory requirements (by a magnitude of 5 and 3 for *minsup* = 0.4% respectively), since the number of frequent continuities grows rapidly as the *minsup* diminishes. ClosedPROWL and ClosedPROWL(Disk) require 129MB and 94MB at the *minsup* = 0.4%, respectively. Note that the number of closed frequent continuities is 3576 at *minsup* = 0.4%. However, the number of closed frequent continuities is very few (1100-1200) between *minsup* = 0.6 and *minsup* = 1.6, so the improvement of memory requirement is only 1-3MB. Thus maintaining closed frequent continuities (*FCTab*) in ClosedPROWL needs 35MB main memory approximately. Meanwhile, we can observe that the pruning strategies of ClosedPROWL increase the efficiency considerably (by a magnitude of 2) through the comparison between ClosedPROWL and COCOA in Fig. 12 (c). In summary, projected window list technique is more efficient and more scalable than Apriori-like, FITI and ComFITI, especially when the number of frequent continuities be- comes really very large.

The advantage of ClosedPROWL over FITI becomes even evident when the maximum window *maxwin* is increased since the number of frequent continuities often increases drastically as *maxwin* increases as shown in Figs. 13 (a) and (b). For the same reason, the same behavior can be observed in Figs. 13 (c) to (h). As shown in these figures, the compression rate varies with various parameter *maxwin*, *I*, *T* and *L*. Practically, it could be related to the characteristics of the data. This could also be phrased as "different data may have different relationships between compression rate and parameters". Note that there is a cross line in Fig. 13 (g), the reason is that the first phase of mining closed frequent itemsets in COCOA and ClosedPROWL requires some extra-cost for closed itemset validation. Thus, for inter-transaction of length one, COCOA and ClosedPROWL not only didn't improve the efficiency, but costs some extra-validation.

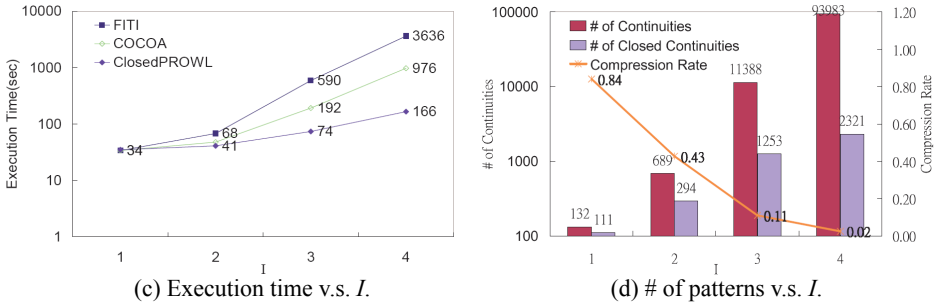
5.2 Real World Dataset

We also apply ClosedPROWL, COCOA and FITI to a data set comprised of ten stocks (electronics industry) in an 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): $\geq 3.5\%$, Upward-Low(UL): $< 3.5\%$ and 0%, Changeless(CL): 0%, Downward-Low(DL): $> -3.5\%$ and $< 0\%$, Downward-High(DH): ≤ -3.5.

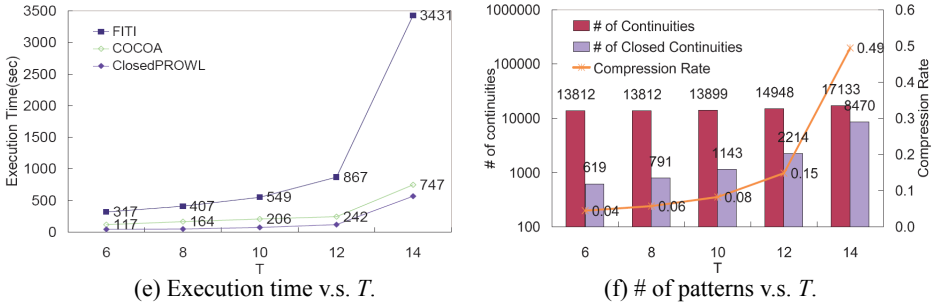
In this case, the average events in each time slot is 10, and the number of events is 50 (10×5). Fig. 14 (a) shows the running time with an increasing support threshold, *minsup*, from 5% to 11%. Fig. 13 (c) shows the same measures with varying *maxwin*. As the *maxwin*/*minsup* threshold increases/decreases, the gap between FITI and ClosedPROWL in the running time becomes more substantial. Finally, Figs. 13 (b) and (d) show the compression rate with varying *minsup* and *maxwin*. As the *maxwin* threshold increases or *minsup* threshold decreases, the compression rate is increased since the number of frequent continuities will increase drastically. Since we only choose ten stocks for this experiment randomly, they could have the same characters and trends. Therefore, the compression rate didn't have a major difference. However, ClosedPROWL still outperforms FITI. The main reason is that the pruning strategies of ClosedPROWL play an important role in efficiency even though compression rate is low.



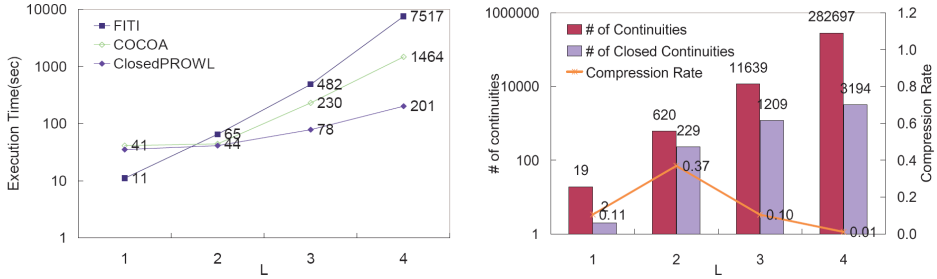
(a) Execution time v.s. *maxwin*. (b) # of patterns v.s. *maxwin*.



(c) Execution time v.s. *I*. (d) # of patterns v.s. *I*.



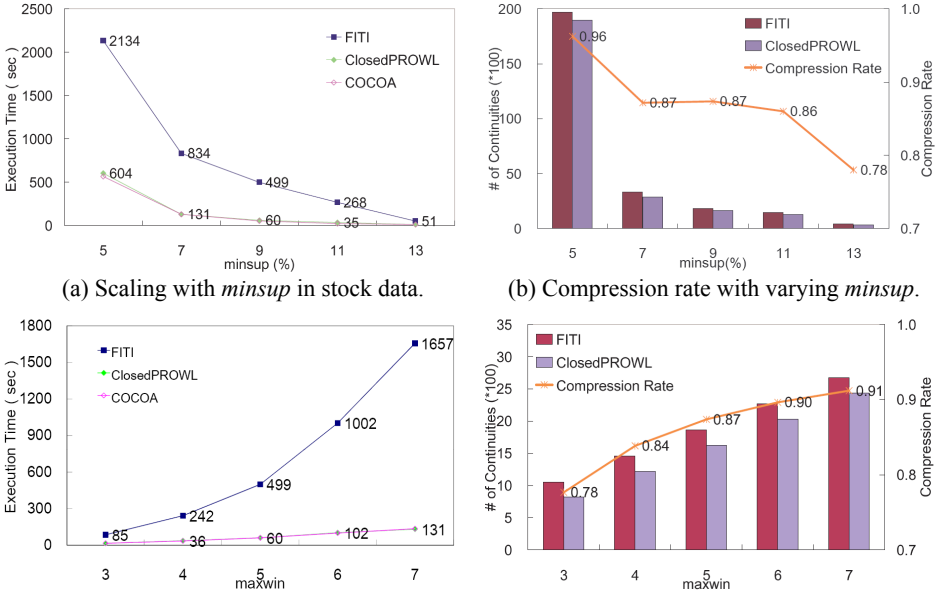
(e) Execution time v.s. *T*. (f) # of patterns v.s. *T*.



(g) Execution time v.s. *L*. (h) # of patterns v.s. *L*.

Fig. 13. Performance comparison II.

In summary, the experiments show the our algorithms are more scalable than FITI in both synthetic and real data. Although ClosedPRWOL needs more space than FITI in pattern maintenance, we also introduce a disk-resident ClosedPRWOL to overcome this problem. Besides, we also use the concept of compressed frequent continuities in FITI, the result demonstrate ComFITI is more efficiency than FITI as our expectation. Furthermore, we also show ClosedPROWL is useful in pattern compression.



(a) Scaling with *minsup* in stock data.

(b) Compression rate with varying *minsup*.

(c) Scaling with *maxwin* in stock data.

(d) Compression rate with varying *maxwin*.

Fig. 14. Performance comparison III.

6. CONCLUSIONS

In this paper, we propose a series of algorithms for the mining of frequent continuities. We show that the three-phase design lets the projected window list technique, which was designed for sequences of events, also applicable to general temporal databases. The proposed algorithm uses both vertical and horizontal database formats to reduce the searching time in the mining process. Therefore, there is no candidate generation and multi-pass database scans. The main reason that projected window list technique outperforms FITI is that it utilizes memory for fast computation. This is the same reason that later algorithms for association rule mining outperform Apriori. Even so, we have demonstrated that the memory usage of our algorithms is actually more compact than the FITI algorithm. Furthermore, with subitemset pruning and sub-continuity checking, Closed-PROWL Successfully discovered efficiently all closed continuities. For future work, maintaining and reusing old patterns for incremental mining is an emerging and important research. Furthermore, using continuities in prediction is also an interesting issue.

REFERENCES

1. R. C. Agarwal, C. C. Aggarwal, and V. Parsad, "A tree projection algorithm for generation of frequent itemsets," *Journal of Parallel and Distributed Computing*, Vol. 61, 2001, pp. 350-371.
2. R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp.

487-499.
3. R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the 11th International Conference on Data Engineering*, 1995, pp. 3-14.
4. M. N. Garofalakis, R. Rastogi, and K. Shim, "Spirit: sequential pattern mining with regular expression of constraints," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 14, 2002, pp. 530-552.
5. J. Han, G. Dong, and Y. Yin, "Efficient mining partial periodic patterns in time series database," in *Proceedings of the 15th International Conference on Data Engineering*, 1999, pp. 106-115.
6. J. Han and J. Pei, "Mining frequent patterns by pattern-growth: methodology and implications," *ACM SIGKDD Explorations – Special Issue on Scalable Data Mining Algorithms*, Vol. 2, 2000, pp. 14-20.
7. 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*, Vol. 8, 2004, pp. 53-87.
8. 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*, 2004, pp. 17-19.
9. K. Y. Huang and C. H. Chang, "Mining periodic pattern in sequence data," in *Proceedings of the 6th International Conference on Data Warehousing and Knowledge Discovery*, 2004, pp. 401-410.
10. 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 the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2004, pp. 509-511.
11. K. Y. Huang, C. H. Chang, and K. Z. Lin, "Prowl: an efficient frequent continuity mining algorithm on event sequences," in *Proceedings of the 6th International Conference on Data Warehousing and Knowledge Discovery*, 2004, pp. 351-360.
12. K. Karimi and H. J. Hamilton, "Distinguishing causal and acausal temporal relations," in *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2003, pp. 234-240.
13. H. Lu, L. Feng, and J. Han, "Beyond intratransaction association analysis: mining multidimensional intertransaction association rules," *ACM Transactions on Information Systems*, Vol. 18, 2000, pp. 423-454.
14. H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in sequences," in *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, 1995, pp. 210-215.
15. H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering generalized episodes using minimal occurrences," in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 146-151.
16. H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, Vol. 1, 1997, pp. 259-289.
17. B. Ozden, S. Ramaswamy, and A. Silberschatz, "Cyclic association rules," in *Proceedings of the 14th International Conference on Data Engineering*, 1998, pp. 412-421.

18. 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*, 2001, pp. 215-226.
19. J. F. Roddick and M. Spiliopoulou, "A survey of temporal knowledge discovery paradigms and methods," *Journal of Parallel and Distributed Computing*, Vol. 14, 2002, pp. 750-767.
20. C. Silverstein, S. Brin, R. Motwani, and J. Ullman, "Scalable techniques for mining causal structures," in *Proceedings of International Conference on Very Large Data Bases*, 1998, pp. 594-605.
21. R. Srikant and R. Agrawal, "Mining sequential patterns: generalizations and performance improvements," in *Proceedings of the 5th International Conference on Extending Database Technology*, LNCS 1057, 1996, pp. 3-17.
22. A. K. H. Tung, H. Lu, J. Han, and L. Feng, "Efficient mining of intertransaction association rules," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, 2003, pp. 43-56.
23. J. Yang, W. Wang, and P. Yu, "Mining asynchronous periodic patterns in time series data," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, 2003, pp. 613-628.
24. M. J. Zaki and C. J. Hsiao, "Charm: an efficient algorithm for closed itemset mining," in *Proceedings of the 2nd SIAM International Conference on Data Mining*, 2002, pp. 457-473.

Kuo-Yu Huang (黃國瑜) received a Ph.D in Computer Science Information Engineering from National Central University in 2006, Jan. He is a senior engineer in the Digital Home Department at CyberLink in Taiwan. His research interests include digital home technique and data mining. He has published over 10 papers on date mining and 10 books on computer programming language.

Chia-Hui Chang (張嘉惠) is an associate professor at National Central University in Taiwan. She received her B.S. in Computer Science and Information Engineering from National Taiwan University, Taiwan in 1993 and Ph.D. in the same department in Jan. 1999. Her research interests include Web information integration, knowledge discovery from databases, machine learning, and data mining.

Kuo-Zui Lin (林國瑞) received his M.S. in Computer Science and Information Engineering from National Central University in 2004. He is an engineer in the Banking Division at Hyweb Tech. in Taiwan. His research interests include pattern discovery and data mining.