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, Chung-Li, Taiwan 320
†{want, kuozui}@db.csie.ncu.edu.tw, ‡chia@csie.ncu.edu.tw

Abstract

Mining frequent patterns in databases is a fundamental and essential 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. Mining compressed/closed frequent patterns 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 on both real world and synthetic datasets shows that our algorithm is more efficient than previously proposed algorithms.

1. Introduction

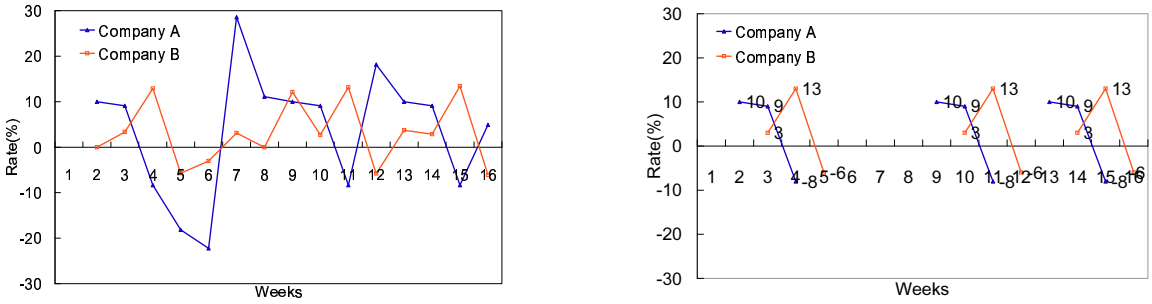
Mining frequent patterns in databases is the fundamental and essential problem for the data mining discipline. Over the past few years, numerous studies have been made in frequent pattern mining, including frequent itemsets [1, 2, 7, 23], sequential patterns [3, 4, 6, 19, 22], frequent episodes [15], periodic patterns [5, 8, 9, 16, 21], frequent continuities [10, 11, 20], causal relations [12, 18], 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 (record) 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 conditional probability (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 association rule R_1 will be "if a customer buys butter, there is 80% confidence that he/she 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/events.

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 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 Tung et al. [20], 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 [13, 14] and periodic patterns [21, 8] 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, Huang et al. 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 and spacial databases. It is assumed that the domain of the dimensional attribute is ordinal and can be divided into equal length intervals, which can be represented by non-negative integers 0, 1, 2, etc. For example, temporal intervals can be divided into days, weeks, months and seasons, etc. and spacial intervals can be divided into miles, regions, latitudes and longitudes, etc. In temporal relationships, we can employ frequent continuities in trend discovery. Trend discovery is applied by comparing the sequence of contiguous data and searching



(a) Price fluctuation rate

(b) Similar shape of the sub-series

Figure 1. Two stock's price fluctuation shape

for similar shapes according to some domain-specific notion of similarity. This type of pattern discovery is used to study problem such as the evolution of stock prices and related populations. For example, Figure 1(a) shows two stock's price fluctuation rates over a 16 weeks. In this case, each transaction (record) in the database registers the weekly price fluctuation rate of two stocks. If we apply the concept of the continuities into trend discovery, a similar shape of the sub-series can be identified explicitly (see Figure 1(b)).

Furthermore, we can extend the concept of the continuity into 2-dimensional remote-sensed images. Take Figure 2 as an example, the database contains 2-dimensional records which describe the locations of buildings. There are four shaded areas indicating that four blocks each contain a bus station, a school and a parking lot. From this pattern, we can discover a rule such as "If a parking lot is located in the north closed area (e.g. Kilometer-Square) of the bus station, there is likely to be a school in the east closed area of the bus station." Thus, we can infer the unknown building at $Location(1,3)$ as school area by the above rule. A direct transformation from 2-dimension data into 1-dimension data is to follow some specific order, such as $T_1 = (1,1)$, $T_2 = (1,2)$, ... $T_{64} = (8,8)$. We then apply a one dimensional continuity mining algorithm in this transformed data. In addition to this application, there are many emerging applications, including key phase extraction in natural language processing, missing value assignment, outlier detection, climatical patterns analysis, telecommunication network fault detection, repeated patterns in biological data, etc.

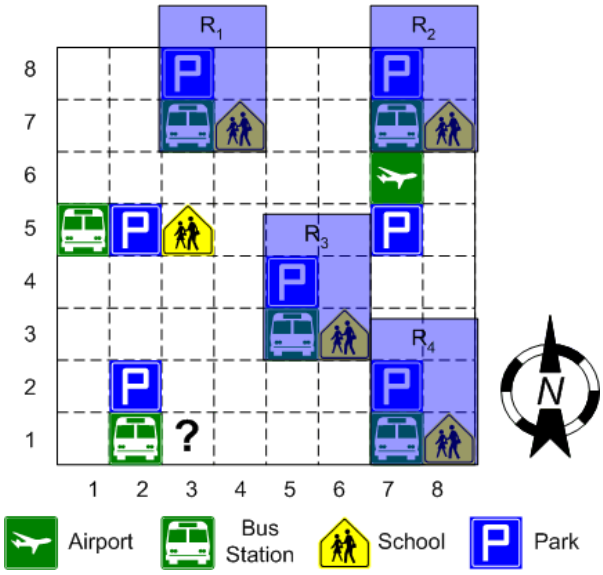


Figure 2. A remote sensed image.

Tung et al. [20] proposed an algorithm, FITI (First Intra Then Inter), for the mining of frequent continuities. FITI is a three-phase algorithm uses the important property "A frequent continuity is composed of frequent intra-transaction itemsets and the don't-care 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 principle to perform a level-wise 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, Huang et al. [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. This reduces not only efficiency but also effectiveness since users have to sift through a large number of mined rules to find useful ones. Therefore, Huang et al. proposed the mining of compressed continuities [10] as an alternative idea to the discovery of frequent continuity.

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 which have no proper super-continuity with the same support in databases. Mining closed frequent continuities 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 our algorithms, including PROWL, COCOA and ClosedPROWL, for mining frequent continuities, compressed continuities and closed frequent continuities respectively. Experiments on both synthetic and real world datasets are reported in Section 5. Finally, conclusions are made in Section 6.

2. Related Works

In this section, we distinguish four pattern mining tasks including sequential patterns, frequent episodes, periodic patterns and frequent continuities and make an overall comparison between frequent itemsets and the four mining tasks.

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 (**G**eneralized **S**equential **P**attern) [19]. However, in situations with prolific frequent patterns, long patterns, or quite low min_sup thresholds, 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 FP-tree [7] and proposed the PrefixSpan algorithm by prefix-projected pattern growth [17] for sequential pattern mining. In addition to algorithms based on horizontal formats, Zaki proposed a vertical-based algorithm SPADE [22]. SPADE utilizes combinatorial properties to decompose the original problem into smaller sub-problems

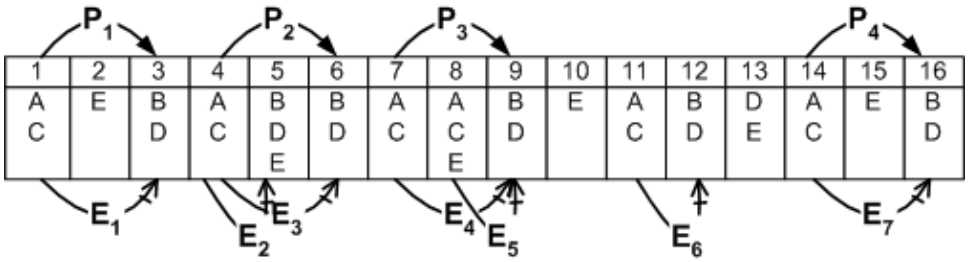


Figure 3. An example of temporal database TD.

that can be independently solved in main-memory using efficient lattice search techniques and simple join operations.

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 [13]. An episode rule is an expression $P[V] \Rightarrow Q[W]$, where P and Q are event sequences and V and W are time window bounds. Mannila et al. defined two kind of episodes: serial and parallel [13]. Serial episodes consider patterns with a specific order, while parallel episodes have no constraints on the relative order of eventsets. Take Figure 3 for example and consider a window size of 3. There are seven matches of the serial episode <AC, BD>, from E_1 to E_7, in the temporal database TD in Figure 3. 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 a framework for discovering frequent episode through a level-wise algorithm, WINEPI [13], for finding all serial/parallel episodes that are frequent enough. This algorithm was an Apriori-like algorithm based on the "anti-monotone" property of episodes. They also presented MINEPI [14], 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.

Unlike episodes, a periodic pattern considers not only the order of events but also the exact positions of events [8, 9, 21]. 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 Figure 3, pattern (AC,*,BD) is a periodic pattern that matches P_1, P_2, and P_3, three contiguous and disjoint

	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	$< I_1, \ldots, I_n >$	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. Comparison of various pattern mining.

matches, where eventset {A,C} (resp. {B,D}) 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 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 [21], 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 Figure 3. The term continuity pattern was coined by Huang et. al. in [11] to replace the general term inter-transaction association defined by Tung, et al. in [20], 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. As noted in the introduction, two algorithms have been proposed for the task. FITI [20] is an Apriori-based algorithm which uses breadth-first enumeration for candidate generation and scans the horizontal-layout database. The PROWL algorithm [11], on the other hand, generates frequent continuities using depth first enumeration and relies on the use of both horizontal and vertical-layout database.

Table 1 shows the comparison of the above mining tasks with frequent itemsets. The column "Order" represents whether the discovered pattern contains 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 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 define the problem of frequent continuity mining. We start from the definition of frequent continuity mining, then introduce the notion of compressed continuity and closed continuity, 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. A temporal database 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. Figure 3). 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 3.1 *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 $2 \leq j \leq W$.*

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. 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 3.2 *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 $(1 \leq j \leq l)$, either p_j = * or $p_j \subseteq d_j$ is true. D is also called a match of P.*

In general, given a temporal database and a pattern P, multiple matches of P may exist. In Figure 3, $P_1, P_2, \ldots, P_4$ are four matches of the continuity pattern [AC,*,BD].

Definition 3.3 *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 3.4 *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 3.5 *Let $|TD|$ be the number of transactions in the temporal database 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*

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

Definition 3.6 *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 3.1 *Let user-specified threshold minimum support ($minsup$) and minimum confidence ($minconf$) be 25 percent and 60 percent respectively. An example of an inter-transaction association rule with maximum time window bound ($maxwin$) = 3 from the database in Figure 3 will be:*

$$[AC, *] \Rightarrow [BD].$$

This rule (Eventset $\{B, D\}$ occurs two slots later after eventset $\{A, C\}$.) holds in the temporal database 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 minimum support level $minsup$ and a maximum time window bound $maxwin$, our task is to mine all frequent continuities from temporal database 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 since users have to sift through a large number of mined rules to find useful ones. Therefore, we proposed an alternative idea to mine "compressed" and "closed" frequent continuities, 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 3.7 *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 Figure 3, pattern [AC,*,D] is a compressed frequent continuity in Figure 3, while pattern [A,*,D], [C,*,D] are not compressed frequent continuity patterns since event A and C are not closed frequent itemsets.

Definition 3.8 *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 $(1 \leq j \leq w)$, $p'_j \subseteq p_{j+o}$ is true for some integer o. The integer o is also called the offset of P and $v + o \leq u$.*

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

With closed frequent continuities, 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 three algorithms including PROWL+, COCOA and ClosedPROWL, for mining frequent continuities, compressed continuities and closed frequent continuities respectively.

4.1 PROWL+

In this section, we extend the PROWL [11] algorithm for frequent continuity mining in temporal databases. Unlike Apriori-like algorithm which uses bread first enumeration for candidate pattern generation, PROWL enumerates new frequent continuities by concatenating a frequent item in the projected window list of an existent frequent continuity using depthth first enumeration. To facilitate this enumeration, PROWL utilizes memory for storing both the time slots for each event (called vertical formats, e.g. Figure 4(a)) and the events at each time slot (called horizontal formats, e.g. Figure 3). To see how PROWL works, we defined the so called projected window list, starting from the definition of the timelist for a 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.

Definition 4.1 *Given a temporal database D and a continuity P with time window bound W, let I_i denote a subsequence of W time slots $I_i = (D[t_i], D[t_i + 1], \ldots, D[t_i + W - 1])$ in D that supports P. Assume there are k matches of P in D. 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 a continuity with window 1. 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 4.2 *Given the time list of a continuity P, $P.timelist$ = $\{t_1, t_2, \ldots, t_k\}$ in the database D, the* **projected window list** *(PWL) of P with offset d is defined as $P.PWL_d$ = $\{w_1, w_2, \ldots, w_k\}$, $w_i = t_i + d$ for $1 \leq i \leq k$. Note that a time slot w_i is removed from the projected list if w_i is greater than $|D|$, i.e. $w_i \leq |D|$ for all i.*

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

Figure 4. Vertical format and frequent itemsets for example database TD

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 $P \cdot X.timelist = P.PWL_1 \bigcap X.timelist$. We call X a **frequent follower** of P. Take Figure 3 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 $P_{[C,*]}.PWL_1$ ($minsup = 25\%$), 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 temporal databases, we utilize the important property that a frequent inter-transaction continuity pattern must be made up of frequent intra-transaction itemsets [20]. 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, Eclat[23], for frequent itemsets mining.

- The second phase is database transformation, where it encodes each frequent itemset (abbreviated

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]

Figure 5. Recovered horizontal database of TD

F.I.) with a unique **ID** and constructs a recovered horizontal database composed of the IDs. To illustrate, Figure 4(b) shows the encoding table of F.I. in Figure 3. Next, based on the time lists of the frequent itemsets together with the encoding table, we construct a recovered horizontal database as shown in Figure 5.

- 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. Figure 6 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 (Step 3∼7 in the $Project$) to find all followers, i.e. IDs. New frequent continuities are formed by concatenating P with a frequent follower (Step 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 three phases of the PROWL+ algorithm using the following example.

Example 4.1 *Given $minsup$ = 25% (4 times) and $maxwin$ = 3, the frequent itemsets for Figure 3 include $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$, $\{A, C\}$ and $\{B, D\}$. Each frequent itemset is encoded with a unique ID as shown in Figure 4(b). Then, we construct a recovered horizontal database composed of the IDs by the time lists of the frequent itemsets (see Figure 5).*

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 Figure 5, 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 simply ignored. Therefore, the frequent continuity generated from prefix $[1]$ is $[[1], [5]]$, i.e. $[A, E]$, with 4 matches. Note that the frequent continuities

Given the recovered horizontal database RD, the encoding table VD, $minsup$, $maxwin$;

Procedure of **PROWL+()**
1. **for each ID** $ID_i \in VD$ **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. $TempVD.clear()$;
3. **for each time instant** $T_i \in TimeList$ **do**
4. **if (**$T_i < |RD|$**) then**
5. $PWL = T_i + 1$;
6. **for each ID** $ID_j \in RD[PWL]$ **do**
7. $TempVD[ID_j].insert(PWL)$;
8. **begin for each ID** $ID_i \in TempVD$ **do**
9. **if (**$TempVD[ID_i].size >= minsup$**) 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**

Figure 6. PROWL+: Frequent Continuity mining algorithm

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. For example, frequent itemsets $\{A\}$ and $\{B\}$ in Figure 4(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 [24] for closed frequent itemset mining. The result of the mining is the compressed continuities, as defined in Definition 3.7. 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 closed continuities which is the target of Closed-PROWL. 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 Figure 4.3. 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.

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 technique can be proven by the following lemma and theorems.

Lemma 4.1 *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 Figure 3 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 4.1 *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.*

Given recovered the horizontal database RD, vertical database VD, $minsup$ and $maxwin$;

Procedure of **ClosedPROWL()**
1. **for each ID** $ID_i \in VD$ **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. $TempVD.clear()$;
3. $PHTab.clear()$; // Hash Table for SubItemsetPruning
4. **for each time instant** $T_i \in TimeList$ **do**
5. **if (**$T_i < |RD|$**) then**
6. $PWL = T_i + 1$;
7. **for each ID** $ID_j \in RD[PWL]$ **do**
8. $TempVD[ID_j].insert(PWL)$;
9. **for each ID** $ID_i \in TempVD$ **do**
10. **if (**$TempVD[ID_i].size >= minsup$**) then**
11. **SubItemsetPruning(**$TempVD[ID_i], PHTab$**);**
12. **for each entity** $H_i \in PHTab$ **do**
13. Pattern[Layer] = $H_i.ID$;
14. **Project(**$TempVD[ID_i], Pattern, Layer + 1$**);**
15. **SubContinuityPruning(**$H_i.Sup, Pattern, Laye + 1$**);**
16. Pattern[Layer] = *;
17. **Project(**$TimeList, Pattern, Layer + 1$**);**
18. **end**

Figure 7. ClosedPROWL: Closed Frequent Continuity mining algorithm

Proof 4.1 *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 4.1), such that $Q \cdot U$ is a super-continuity of $P \cdot U$, and $(P \cdot U).timelist = P.PWL_{|U|} \bigcap U.timelist = Q.PWL_{|U|} \bigcap U.timelist = (Q \cdot U).timelist$. Therefore, $P \cdot U$ is not a closed continuity.*

Theorem 4.2 *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 4.2 *Consider the continuity $U = [p_1,\ p_2,\ \ldots,\ p_w,\ p_{w+1} \cup\ p'_{w+1}]$. $U.timelist = P.timelist \bigcap Q.timelist$. Since $P.timelist = Q.timelist$, we have $U.timelist = P.timelist = Q.timelist$. Using Theorem 8, 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 Figure 8, 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 $H_i -> size\%PHSize$ (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 (line 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. (line 8–10). To apply property 3, we remove both $H_i.ID$ and $H_j.ID$ from $PHTab[bkNum]$ if their timelists are equivalent (line 11–14).

The sub-itemset pruning technique removes the non-closed sub-continuity of closed frequent continuities with zero offset since the pruning is invoked within a local search of a continuity. For those sub-continuities of closed frequent continuities with non-zero offset, they can still be generated in the mining process. Therefore, we need a checking step to remove non-closed continuities. Again, a hash structure, FCTab (frequent continuity table), is devised to facilitate efficient sub-continuity checking

SubProcedure of **SubItemsetPruning**($TempVD[ID_i], PHTab$)
1. bkNum = $TempVD[ID_i].size\%BucketSize$;
2. IsClosed = true;
3. **for each entry** $H_j \in PHTab[bkNum]$ **do**
4. **if (**$TempVD[ID_i].size == H_j.sup$ **) then**
5. **if (**$Pattern(ID_i) \subset Pattern(H_j.ID)$ **) then**
6. IsClosed = false; // Prune Subtree of ID_i
7. break;
8. **else if (**$Pattern(ID_i) \supset Pattern(H_j.ID)$ **) 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 $PHTab[bkNum]$;

Figure 8. SubItemsetPruning: Sub-Itemset Pruning strategy

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

Figure 9. SubContinuityPruning: Sub-Continuity Pruning strategy

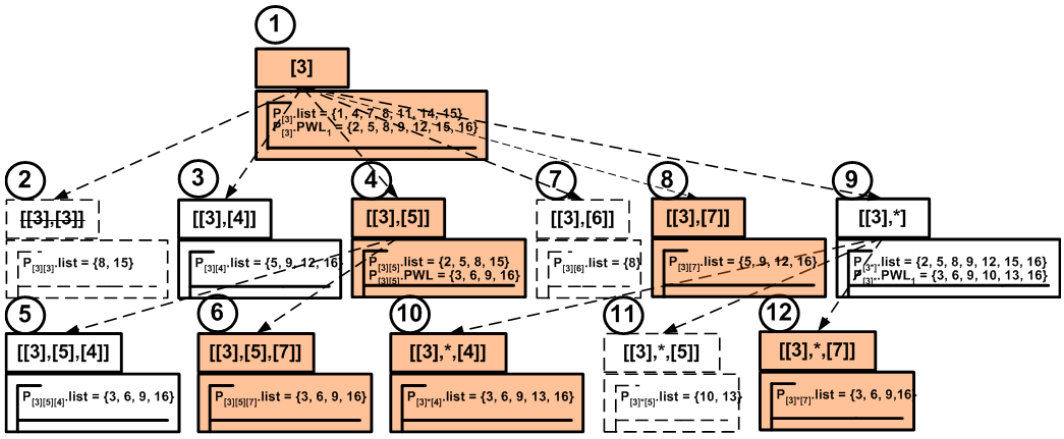


Figure 10. Process of ClosedPROWL for prefix [3].

using the following as the hashing function:

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

The procedure SubContinuityPruning is shown in Figure 9. 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 (line 5–7). On the other hand, if X is a sub-continuity of an existing one Y in $FCTab$, we simply ignore X (line 8–10). We will use the following example to illustrate the mining process of ClosedPROWL and its corresponding flowchart is depicted in Figure 10.

Example 4.2 *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 Figure 5(b). The mining process for prefix* [3] *is shown in Figure 10. 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*

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 $[C, E]$

Figure 11. Pruning header table (PHTab) for sub-itemset pruning.

continuity $[[3], [4]]$ *is removed (see Figure 11(a)). Therefore, only the continuities* $[[3], [5]]$ *and* $[[3], [7]]$ *are inserted into* $FCTab$*.*

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 Figure 11(b)) because* $[4] \subset [7]$ *(*$\{D\} \subset \{B, D\}$*). 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 Figure 10. Finally, we apply the sub-continuity pruning to remove non-closed continuities, the frequent continuity table after mining in prefix* $[3]$ *is shown as Figure 12. 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 temporal database* TD *in Figure 3.*

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.

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

Figure 12. Frequent continuity table (FCTab) for sub-continuity checking.

Lemma 4.2 *The time list of a continuity* $P = [p_1, p_2, ..., p_w]$ *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 4.3 *A closed continuity is composed of only closed itemsets and don't care characters.*

Proof 4.3 *Assume* $P = [p_1, p_2, \ldots, p_W]$ *is a closed continuity, and some of the* p_i*s 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 4.2). 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 4.3 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 4.4 *The ClosedPROWL algorithm generates all closed frequent continuities.*

Proof 4.4 *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 4.3, 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-itemet pruning only removed non-closed continuities (Theorem 4.1).*

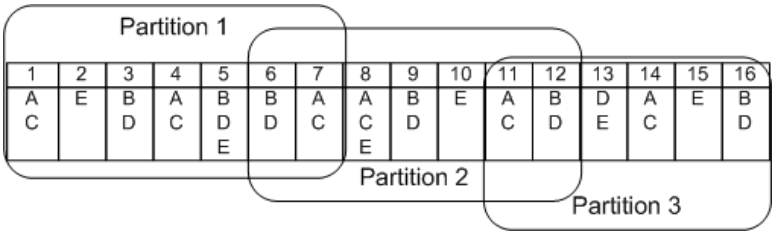


Figure 13. Partition example.

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 temporal database 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. The local frequent continuities are false-positive continuities. Therefore, an additional scan of the original database is necessary in order to determine the global frequent continuities. Take Figure 13 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 Figure 13. Firstly, we mine the local frequent continuities which satisfied the local minimum support 25% ($\lceil 7 * 25\% \rceil = 2$ times) in each partition. Finally, we scan the temporal database once to check which continuity is true-positive frequent continuities, satisfying the global minimum support of 25% ($\lceil 16 * 25\% \rceil = 4$ times).

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 $n * t$ (n entities in $PHTab$). Similarly, the space requirement in $FCTab$ is $m * t$, 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 continuitities. 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.

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

Table 2. Comparison of various mining tasks

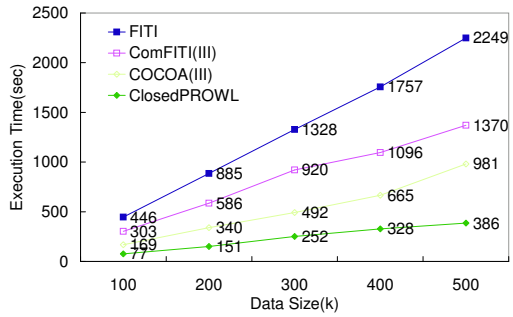
Sym	Definition	Default
$\|D\|$	# of time instants	100K
N	# of events	500
T	Average transaction size	10
$\|C\|$	# of candidate continuities	2
L	Average continuity length	3
I	Average itemset length	2
W	Average window length	5
Sup	Average support	2%

Table 3. Meanings of symbols

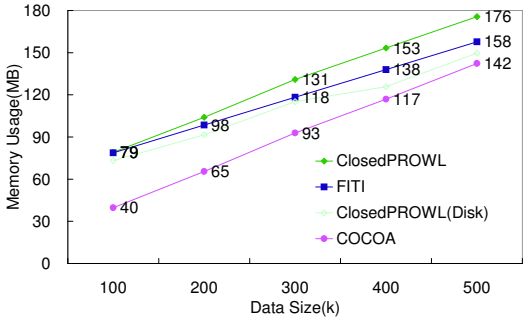
5.1 Synthetic data

For performance evaluation, we use synthetically generated temporal data, D, consisting of N distinct symbols and $|D|$ time instants. A set of candidate 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 $Sup * |D|$. 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.

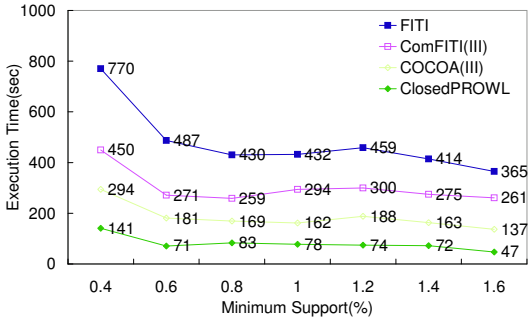
We start by looking at the performance of ClosedPROWL with default parameter $minsup = 0.6\%$ and $maxwin = 5$. Figure 14(a) shows the scalability of the algorithms with varying database size. ClosedPROWL is faster than FITI (by a magnitude of 5 for $|D| = 500K$). The scaling with database size was linear. Therefore, the scalability of the projected window lists technique is proved. Another



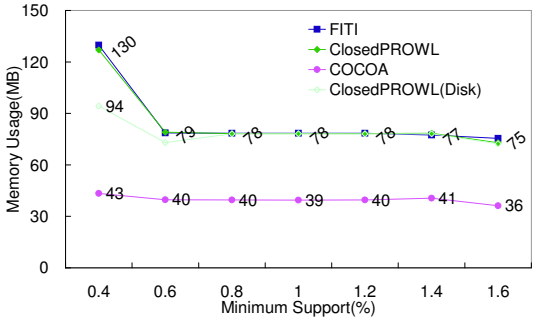
(a) Execution Time v.s. Data Size



(b) Memory Usage v.s. Data Size



(c) Execution Time v.s. $minsup$



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

Figure 14. Performance comparison I

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 Figure 14(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 $|D| = 500K$, with an original database of 12.2 MB. Since ClosedPROWL requires additional memory to maintain frequent continuities ($FCTab$), we modify the algorithm as described in Section 4.6 to disk-resident ClosedPROWL (labelled ClosedPROWL(Disk)). As illustrated in Figure 14(b), the memory requirement of the ClosedPROWL(Disk) is thus less than FITI but more than COCOA for subitemset pruning ($PHTab$).

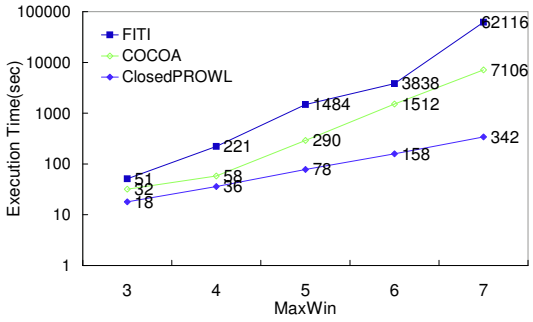
The runtime and memory usage of FITI and ClosedPROWL on the default data set with varying min-

imum support threshold, $minsup$, from 0.4% to 1.6% are shown in Figures 14(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 $minsup$ diminishs. ClosedPROWL and ClosedPROWL(Disk) require 129MB and 94MB at the $minsup = 0.4\%$, respectively. 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 Figure 14(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 becomes really very large.

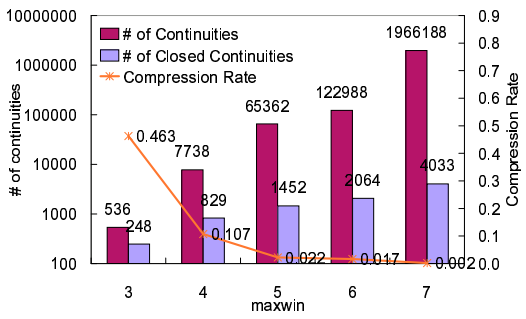
The advantage of ClosedPROWL over FITI becomes even evident when the maximum window $maxwin$ is increased since the number of frequent continuities often increases drastically as $maxwin$ increases as shown in Figure 15 (a) and (b). For the same reason, the same behavior can be observed in Figures 15 (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".

5.2 Real World Dataset

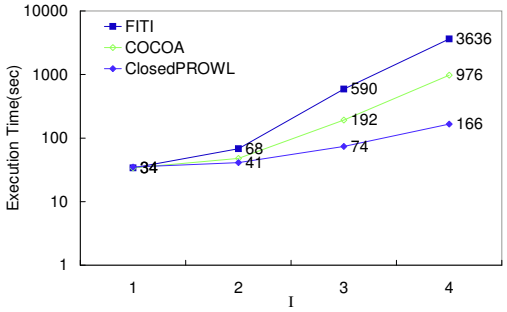
We also apply ClosedPROWL, COCOA and FITI to a data set comprised of ten stocks (electronics industry) in an the Taiwan Stock Exchange Daily Official list for 2618 trading days from September 5, 1994 to June 21, 2004. We discretize the stock price of go-up/go-down into five level: upward-high(UH): $>= 3.5\%$, 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). Figure 16(a) shows the running time with an increasing support threshold, $minsup$, from 5% to 11%. Figure 15(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, Figures 15(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.



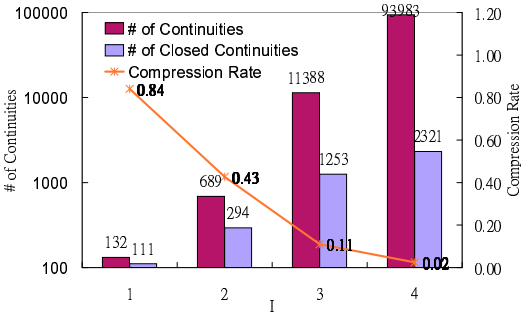
(a) Execution Time v.s. $maxwin$



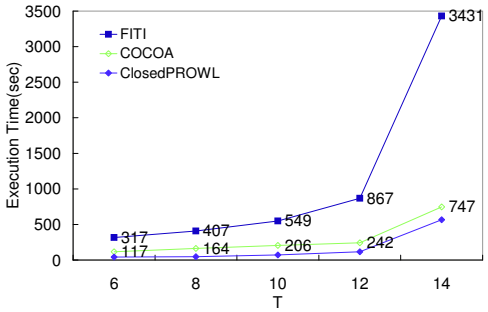
(b) # of patterns v.s. $maxwin$



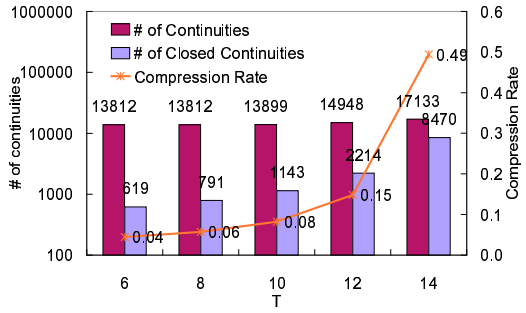
(c) Execution Time v.s. I



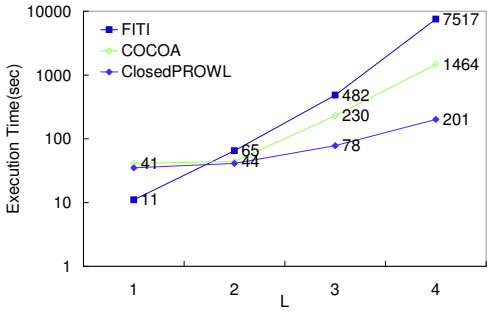
(d) # of patterns v.s. I



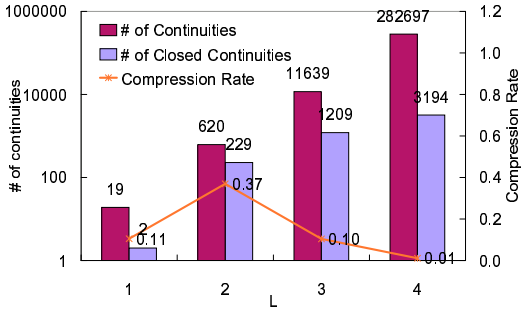
(e) Execution Time v.s. T



(f) # of patterns v.s. T

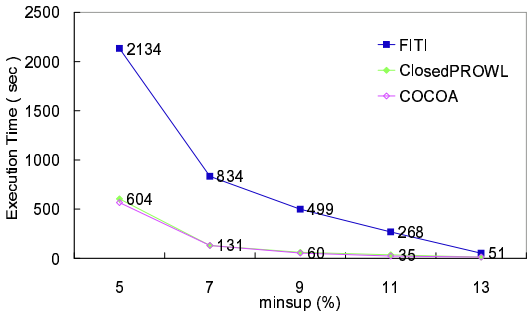


(g) Execution Time v.s. L

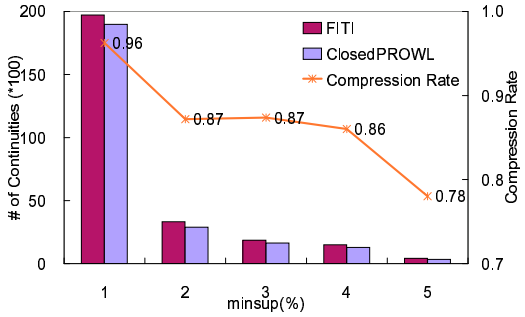


(h) # of patterns v.s. L

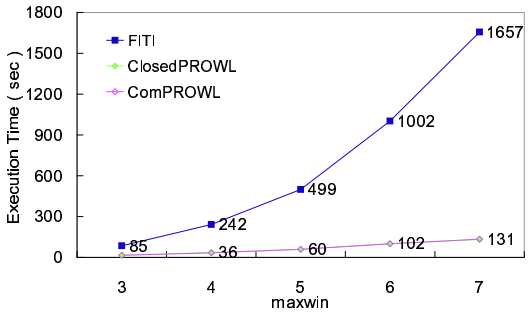
Figure 15. Performance comparison II



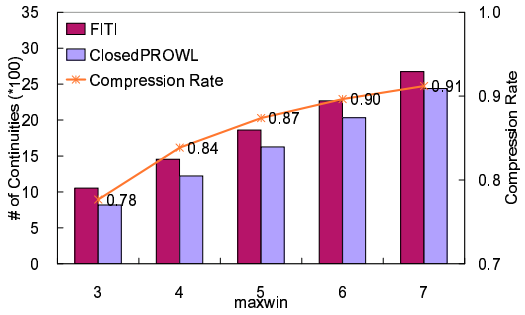
(a) Scaling with $minsup$ in stock data



(b) Compression Rate with varying $minsup$



(c) Scaling with $maxwin$ in stock data



(d) Compression Rate with varying $maxwin$

Figure 16. Performance comparison III

Finally, we apply ClosedPROWL to protein sequences to discover tandem repeats, which is an important problem in bioinformatics. We used data from the PROSITE database of the ExPASy Molecular Biology Server (http://www.expasy.org/). We selected a protein sequence P13813 (110K_PLAKN) with a known tandem repeats "{E,E,T,Q,K,T,V,E,P,E,Q,T}". As expected, several closed frequent continuities which are related to the known tandem repeat are discovered. For example, we found continuities: {E,E,T,Q,*,T,V,E,P,E, Q,*}, {E,E,T,Q,*,T,V,E,P,E,*,T,}, {E,E,*,Q,K,T,V,E,P,E,Q,*} and {E, E,*,Q,*,T,V,E,P,E,Q,T,}, which have 2 mismatches of the known tandem repeat. This indicates that continuity mining can be used in protein sequence mining.

6. Conclusion

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 the same reason that later algorithms for association rule mining outperform Apriori. Even so, we have demonstrated that the memory usage of our algorithms are actually more compact than the FITI algorithm. Furthermore, with subitemset pruning and sub-continuity checking, ClosedPROWL 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. *In Journal of Parallel and Distributed Computing*, 61(3):350–371, 2001.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 487–499, 1994.

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

[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 (TKDE)*, 14(3):530–552, 2002.

[5] 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.

[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)*, 2(2):14–20, 2000.

[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 (DMKD)*, 8(1):53–87, 2004.

[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 (DBA'04)*, pages 17–19, 2004.

[9] 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)*, 2004. To Appear.

[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 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'04)*, 2004. To Appear.

[11] 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)*, 2004. To Appear.

[12] K. Karimi and H. J. Hamilton. Distinguishing causal and acausal temporal relations. In *Proceeding s of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'03)*, pages 234–240, 2003.

[13] 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.

[14] H. Mannila, H. Toivonen, and A. I. Verkamo. 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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 (VLDB'98)*, pages 594–605, 1998.

[19] 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.

[20] 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.

[21] 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.

[22] M. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.

[23] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 12(3):372–390, 2000.

[24] M. J. Zaki and C. J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proceedings of 2nd SIAM International Conference on Data Mining (SIAM 02)*, pages 457–473, 2002.