PROWL: An Efficient Frequent Continuity Mining Algorithm on Event Sequences

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 association rule in event sequences is an important data mining problem with many applications. Most of previous studies on association rules are on mining intra-transaction association, which consider only relationship among the item in the same transaction. However, intra-transaction association rules are not a suitable for trend prediction. Therefore, inter-transaction association is introduced, which consider the relationship among itemset of multiple time instants. In this paper, we present PROWL, an efficient algorithm for mining inter-transaction rules. By using projected window method and depth first enumeration approach, we can discover all frequent patterns quickly. Finally, an extensive experimental evaluation on a number of real and synthetic database shows that PROWL significantly outperforms previous method.

1 Introduction

Mining frequent patterns in event sequences is a fundamental problem in data mining areas. There are many emerging applications in pattern mining, including stock market price movement, telecommunication network fault analysis and web usage recommendation. Therefore, there are various directions in pattern mining, such as frequent itemset, sequential pattern, frequent episode [2], periodic pattern [1] etc. Some of them are on mining intra-transaction association rules like "The price of 3Com(COMS) and Cisco(CSCO) always go up together on the same day with 70% confidence". Such information can be used for sale purposes such as decision making because the co-occurrence patterns of records can be used to infer another record. Despite the above rule reflects some relationship among the prices, it is not a suitable and definite rule for trend prediction. The investors may be more interested in a rule like "When the price of COMS goes up, the price of CSCO will go up with 60% probability three days later." Therefore, we need a pattern that shows the relationships between items in different transaction records. In order to classify these two rules explicitly, we call the former rule as **intra-transaction associations**, the latter rule as **inter-transaction associations**.

In addition to inter-transaction association rules, there are two other kinds of association mining from different transaction records, frequent episodes and periodic patterns. An episode is defined to be a collection of events in a specific

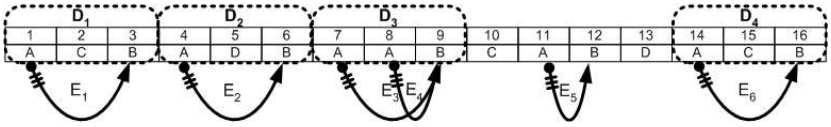

Fig. 1. Event Sequence S.

window interval that occur relatively close to each other in a given partial order [2]. Take Figure 1 as an example, there are six matches of episode $\{A, B\}$, from E_1 to E_6, in event sequence S. Mannila et al. proposed an algorithm, WINEPI [2], for finding all episodes that are frequent enough. This algorithm was an Apriori-like algorithm based on the "anti-monotone" property of episodes. They also presented MINEPI, an alternative approach, to the discovery of frequent episodes based on minimal occurrences of episodes. Note that an episode considers only the partial order relation, instead of the actual positions, of events in a window.

Unlike episodes, a periodic pattern [1] considered not only the order of events but also the exact positions of events. To form periodicity, a list of k disjoint matches is required to form a contiguous subsequence with k satisfying some predefined min_rep threshold. As illustrated in Figure 1, pattern $\{A, *, B\}$ is a periodic pattern that matches D_1, D_2, and D_3, three continuous while disjoint matches, where A (resp. B) occurs at the first (resp. *third*) position of each match. The symbol "*" (the "don't care" position in a pattern) is introduced to allow partial periodicity. Sometimes, we use a 4-tuple (P, l, *rep*, *start*) to denote a segment of pattern P with period l starting from position *start* for *rep* times. In this case, the segment can be represented by ($\{A, *, B\}$, 3, 3, 1).

The concept and definition of inter-transaction association rules are first introduced in [4] by Tung et al. A frequent pattern in inter-transaction association rule is similar to a periodic pattern, but without the constraint on the contiguous and disjoint matches. To distinguish the frequent patterns in inter-transaction association rules from the frequent patterns in intra-transaction association rules, we use the term "**frequent continuities**" for the frequent patterns in inter-transaction association rules. Let us return to our previous example in Figure 1. $\{A, *, B\}$ is a continuity with four matches D_1, D_2, D_3, and D_4.

Inter-transaction association rules can be mined by the FITI algorithm proposed in [3]. FITI consists of two phases: intra-transaction and inter-transaction itemsets mining. Similar to Apriori-like algorithms, FITI could generate a huge number of candidates and require several scans over the whole data sequence to check which candidates are frequent. To illustrate, if there are I frequent 1-patterns, the FITI algorithm will generate approximately I^W 2-pattern candidates for window size W. Experimentally, the bottleneck of the FITI method comes from its step-wise candidate pattern generation and test. Therefore, it is desirable to develop an algorithm which avoid candidate generation and reduce the search space. With this motivation, we devised a two-phase algorithm, PROWL, for mining frequent continuities from event sequences. We introduce a

projected window mechanism and employ depth first enumeration approach to discover all frequent continuities.

The remaining parts of the paper are organized as follows. We define the problem of frequent continuity mining from event sequence in Section 2. Section 3 presents our algorithm for mining frequent continuity in event sequence. Experiments on both synthetic and real world data are reported in Section 4. Finally, conclusion is made in Section 5.

2 Problem Definition

In this section, we define the problem of frequent continuity mining. The problem definition is similar to [3] but is restricted to a sequence of events. Let $E = (e_1, e_2, \ldots, e_n)$ be a set of literals, called events. A event sequence S is a set of time records where each time record is a tuple (tid, x_i) for time instant tid and event x_i ($x_i \in E$). A sequence stored in form of (tid, x_i) is called horizontal format (e.g. Figure 1).

Definition 1. *A continuity pattern (or a continuity in short) with window W is a nonempty sequence $P = (p_1, p_2, \ldots, p_W)$ where p_1 is an event and others are either an event or *, i.e. $p_j \in E$ 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 pattern can start anywhere in a sequence, we only need to consider patterns that start with a non-"*" symbol. A continuity P is called an i-continuity or has length i if exactly i positions in P contain event. For example, $\{A, *, *\}$ is a 1-continuity; $\{A, *, C\}$ is a 2-continuity which has length 2.

Definition 2. *Given a continuity pattern $P = (p_1, p_2, \ldots, p_W)$ and a subsequence of W slots $D = (d_1, d_2, \ldots, d_W)$ in S, we say that P* **matches** *D (or D supports P) if and only if, for each position j ($1 \leq j \leq l$), either $p_j = *$ or $p_j = d_j$ is true. D is also called a match of P.*

In general, given a sequence of events and a pattern P, multiple matches of P may exist. In Figure 1, $D_1, D_2, \ldots, D_4$ are four matches of pattern $\{A, *, B\}$.

Definition 3. *An* **inter-transaction association rule** *is an implication of the form $X \Rightarrow Y$, where*

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

Similar to the studies in mining intra-transaction rules, we also introduce two measures of inter-transaction association rules: support and confidence.

[1] The **concatenation** of two continuity patterns $P = (p_1, \ldots, p_{w_1})$ and $Q = (q_1, \ldots, q_{w_2})$ is defined as $P \cdot Q = (p_1, \ldots, p_{w_1}, q_1, \ldots, q_{w_2})$.

Definition 4. *Let $|S|$ be the number of transactions in the event sequence S. 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*

$$support = \frac{Sup(X \cdot Y)}{|S|},\ confidence = \frac{Sup(X \cdot Y)}{Sup(X)}. \quad (1)$$

The problem is formulated as follows: given a minimum support level $minsup$ and a minimum confidence level $minconf$, our task is to mine the complete set of inter-transaction association rules from an event sequence with support greater than $minsup$ and confidence greater than $minconf$. We illustrate the concepts with an example. Let $minsup$ and $minconf$ be 25% and 60% respectively. An example of an inter-transaction association rule from the event sequence in Figure 1 will be: $\{A, *\} \Rightarrow \{B\}$. This rule (Event B occurs two slots later after event A.) holds in the sequence S with support 25% and confidence 67%.

3 The PROWL Algorithm

In this section, we explore methods for mining frequent continuities in an event sequence. Our algorithm, PROWL (PROjected Window Lists), uses a recursive depth first enumeration strategy to discover all frequent continuities from an event sequence. To avoid candidate generation, we use a vertical data format together with a horizontal format for efficient continuity generation. Table 1 shows the vertical format for the event sequence S in Figure 1, where a time list is maintained for each event. A time list of an event records the time slots where the event occurs in the sequence.

Definition 5. *Given a sequence of events S and a continuity P with window W, let I_i denotes a subsequence of W time slots $I_i = (S[s_i], S[s_i+1], \ldots, S[s_i+W-1]$ in S that supports P. Suppose there are k matches of P in S. The time list of P is defined as $P.list = \{s_1 + W - 1, s_2 + W - 1, \ldots, s_k + W - 1\}$.*

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

Definition 6. *Projected Window List (PWL): Given a time list of a continuity P, $P.list = \{o_1, o_2, \ldots, o_k\}$ in the event sequence S, the projected window list of P is defined as $P.PWL = \{w_1, w_2, \ldots, w_k\}$, $w_i = o_i + 1$ 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 $|S|$, i.e. $w_i \leq |S|$ for all i. If an event X is frequent in the projected window list of pattern P, we refer to the concatenation $P \cdot X$ as an extension of P.*

The PROWL algorithm mines frequent continuities by the following phases.

- Initial phase: The sequence S is first read into memory and scanned once. For each event, a time list is maintained to record its occurring time slot. The number of occurrences is also accumulated for frequent event filtering.

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

Table 1. Vertical format of the event sequence S in Figure 1

– Recursive phase: For each frequent 1-continuity, we calculate a projected window list (PWL) from the pattern's time list and find frequent events in its PWL. We then output the frequent continuity formed by current pattern and the frequent events in the PWL. For each extension pattern, the process is applied recursively to find all frequent continuities until the projected window list becomes empty or the window of a continuity is greater than the maximum window.

3.1 An Example

The PROWL algorithm can be best understood by an illustrative example described below and its corresponding flowchart is depicted in Figure 2.

Example 1. Given $Sup = 3$ and $maxwin = 4$, the frequent events for Table 1 include A, B and C. For frequent 1-continuity $\{A\}$, the projected window list is $P_A.PWL = \{2, 5, 8, 9, 12, 15\}$. Note that $P_A.PWL$ is also the time list of continuity $\{A, *\}$. By examining the time slots of $P_A.PWL$ in Figure 1, all the continuities with window 2 having prefix $\{A\}$ can be generated by concatenating $\{A\}$ with a frequent event in $P_A.PWL$ or the don't care symbol. For instance, the corresponding events for the time slots in $P_A.PWL$ are C, D, A, B, B, C, respectively. For each event E_i, a time list is constructed accordingly. Since D is not frequent in S, it is simply ignored. Furthermore, as there are no frequent events in $P_A.PWL$, the only frequent continuity generated from $\{A\}$ is $\{A, *\}$ (with 6 matches).

Recursively, we apply the above process to continuity $\{A, *\}$. The projected window list of $\{A, *\}$ is $P_{\{A,*\}}.PWL = \{3, 6, 9, 10, 13, 16\}$. In this layer, we find a frequent event B in time records of $P_{\{A,*\}}.PWL$. Thus, two continuities: $\{A, *, B\}$ and $\{A, *, *\}$ are generated with time list $P_{\{A,*,B\}}.list = \{3, 6, 9, 16\}$ and $P_{\{A,*,*\}}.list = \{3, 6, 9, 10, 13, 16\}$, respectively. The extensions of the continuities can be mined by applying the above process respectively to each continuity as shown in Figure 2. Note that the projected window list of $\{A, *, B\}$ is $\{4, 7, 10\}$, because time record 17(16+1) is greater than sequence length 16. Similarly, we can find all frequent continuities having prefix $\{B\}$, respectively, by constructing $P_B.PWL$ and mining them respectively. The set of frequent continuities is the collection of patterns found in the above recursive mining process.

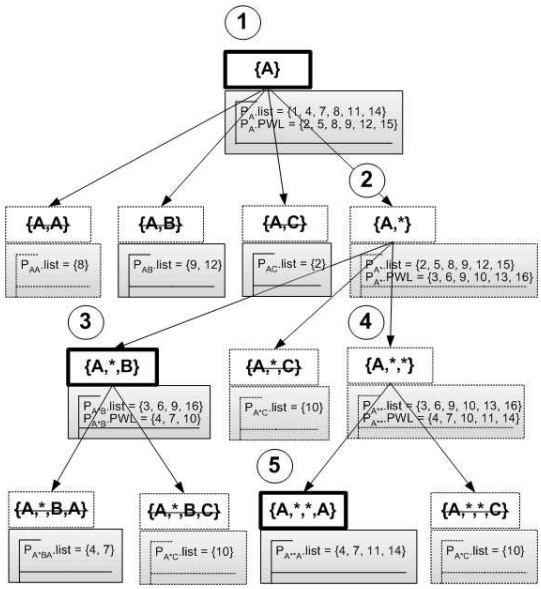

Fig. 2. The overall process of PROWL for Figure 1.

3.2 The PROWL algorithm

The main idea of PROWL is to utilize the memory for both the event sequence and the indices in the mining process. The event sequence S is read once from disk and each event is associated with a time list containing indices to the time slots where it occurs. Based on the concepts of projected window list and "anti-monotone" property, we can generate all frequent continuities by depth-first enumeration. PROWL discovers all frequent continuities recursively by searching the time slots in the projected window lists immediately. Figure 3 outlines the proposed PROWL algorithm.

In the first phase, we scan event sequence S once and transform S into vertical format (Step 1∼2 in the $Prowl$). To reduce the number of events for enumeration, the number of occurrences for each event is accumulated during sequence reading and non-frequent events are masked (Step 3∼5 of $Prowl$). Therefore, non-frequent events will not be enumerated in the recursive phase. In the second phase, the procedure $Project$ is applied recursively to enumerate all continuities with known frequent continuities as their prefixes. For each frequent continuity P, we transform its time list into a projected window list and examing the time slots of it's projected window in S (Step 4∼8 in the $Project$). Note that the horizontal format of sequence S is maintained in main memory for fast access to the event at each time slot $o_i \in P.PWL$. The function $TempEvent(e).insert(o_i)$ insert time slot o_i into the time list of event e. If an event E_i is frequent, the continuity $P \cdot E_i$ is output (Step 10∼13 of $Project$). The recursive call stops when the layer is greater than $maxwin$ (Step 1 of $Project$).

Given event sequence S, Sup, $maxwin$;

Procedure of **Prowl()**
1. **for** $i = 1$ **to** $|S|$ **do**
2. $EventSet[S[i]].insert(i)$;
3. **for each event** $E_i \in EventSet$ **do**
4. **if** ($EventSet[E_i].size < Sup$) **then**
5. **for each time instant** $o_i \in EventSet[E_i]$ **do**
6. $S[o_i] = *$;
7. **for each event** $E_i \in EventSet$ **do**
8. **if** ($EventSet[E_i].size >= Sup$) **then**
9. Pattern[0] = E_i;
10. **for** $j = 1$ **to** $maxwin - 1$ **do**
11. Pattern[j] = *;
12. **Project**($EventSet[E_i], Pattern, 1$);
13. **end**

Subprocedure of **Project**($TimeList$, $Pattern$, $Layer$)
1. **begin if** ($Layer <= maxwin$) **then**
2. $PWL = NULL$;
3. $TempEvent = NULL$;
4. **for each time instant** $T_i \in TimeList$ **do**
5. **if** ($T_i < |S|$) **then**
6. $PWL.insert(T_i + 1)$;
7. **for each time instant** $o_i \in PWL$ **do**
8. **if** $S[o_i] \neq *$ **then**
9. $TempEvent[S[o_i]].insert(o_i)$;
10. **begin for each event** $E_i \in TempEvent$ **do**
11. **if** ($TempEvent[E_i].size >= Sup$) **then**
12. Pattern[Layer] = E_i;
13. **Project**($TempEvent[E_i], Pattern, Layer + 1$);
14. Output Pattern;
15. Pattern[Layer] = *;
16. **Project**($TempEvent[E_i], Pattern, Layer + 1$);
17. **end**
18. **end**

Fig. 3. PROWL: Frequent Continuity mining algorithm

In contrast with Apriori-like algorithms, we only generate longer pattern by shorter ones. It does not generate any candidate patterns for checking. Besides, the projected window lists are actually smaller than the original ones. The situation will be illustrated in the scale-up experiments discussed later.

4 Experimental Result

In this section, we report a performance study of the algorithm proposed in this paper and applications of frequent continuity in real world data. We first investigate the performance of PROWL and compare the result with the FITI algorithm proposed in [3] using synthetic data. The PROWL algorithm is then applied to real world data for frequent continuity mining.

4.1 Synthetic data

To obtain reliable experimental results, the method to generate synthetic data we employed in this study is similar to the ones used in prior works [1]. We use a synthetically generated event sequence consisting of $|N|$ distinct symbols and $|S|$ events. A set of candidate continuities C is generated as follows. First, we decide the window length of a continuity from a geometrical distribution with mean W. Then L $(1 < L < W)$ positions are chosen randomly for non-empty events. The number of occurrences of a continuity follows a normal distribution with mean Avg_Sup. The continuity is then inserted into the sequence Avg_Sup times with gap between W to $2W$. A total of $|C|$ complex patterns are generated. After all candidate patterns are generated, events are picked at random from the symbol set N for empty time slots. The default parameter is S20K-N1K-C10-L4-W10-Avg_Sup=0.5. The experiments are conducted on a computer with a CPU clock rate of 1G MHz and 1.5G MB of main memory, the program is written by visual C++ in windows 2000 platform.

4.2 Comparison of PROWL with FITI

We reported experimental results on the default data set. For comparison with PROWL, we implement FITI algorithm which is proposed in [3]. Both PROWL and FITI algorithms show linear scalability with the size of a sequence from 10K to 80K as shown in Figure 4(a). However, PROWL is much more scalable than FITI. As the size of a sequence grows up, the difference between the two methods becomes larger and larger.

Figure 4(b) shows that the running time of PROWL and FITI with variable number of candidates. It shows that the running time of PROWL increases smoothly while FITI increases exponentially as the number of candidates increases. To test the scalability with the pattern length, we also execute an experiment with varying pattern length. The results are presented in Figure 4(c). Overall, PROWL is about an order of magnitude faster than FITI. This is because the large number of candidates that need to be generated and tested in

FITI as the pattern length grows. Figure 4(d) shows the scalability of the algorithm over the varying window size. Note that the parameter *maxwin* is set to the window size W of dataset. As the window size becomes larger, more candidates need to be generated in FITI. On the contrary, PROWL doesn't generate any candidate for checking. Thus, for a fixed pattern length, the running time of PROWL is smooth with the varying window size.

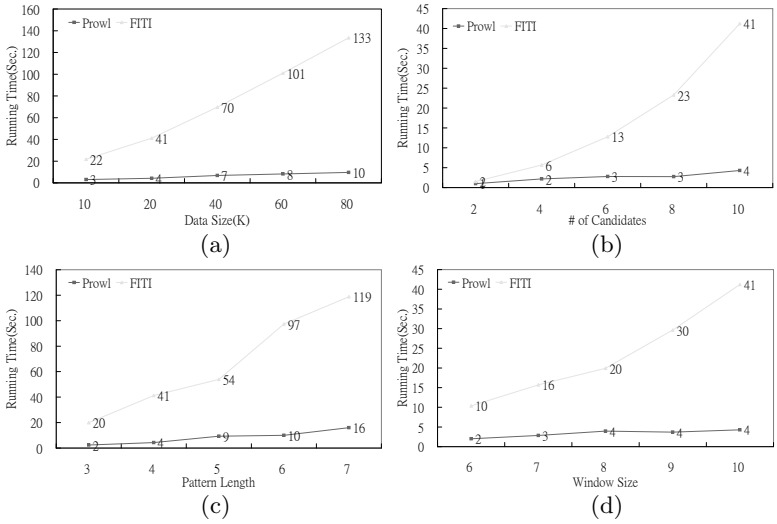

Fig. 4. Scale-up performances with (a) sequence size (b) number of candidates (c) pattern length (d) window size.

4.3 Real World Data

We also run PROWL on a variety of different real world data sets to get a better view of the usefulness of frequent continuities in event sequences. The data sets are taken from the UNIX user usage logs in UCI Machine Learning Database Repository. The UNIX user usage logs contains 9 subsets of sanitized user data drawn from the command histories of 8 UNIX computer users at Purdue over the course of up to 2 years. We show only the data of User0, User1 and User2 due to space limitation. The description of the data used, the parameters, and the number of frequent continuities discovered are presented in Table 2.

Finally, we apply PROWL to protein sequences to discover tandem repeats, which is an important problem in bioinformatics. We used data in 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 continuities which are related to the known tandem repeat are discovered. It is indicated that our algorithm can be used in protein sequence mining.

Data Set	Events	Event Types	Support	Maximum Window	# of continuity
UNIX User0	8974	197	100	10	341
UNIX User1	19881	288	200	10	711
UNIX User2	18738	310	200	10	2183
Protein Sequence	296	22	10	12	10165

Table 2. Dataset Characteristics

5 Conclusion

In this paper, we proposed an algorithm, PROWL, for mining frequent continuities in event sequence. The idea is to utilize memory for storing event sequence in both horizontal and vertical data format. PROWL generates frequent continuities by applying a projected window method recursively. The experiments show that the method is efficient and flexible. We compared PROWL with previous research, and the result reported that Prowl outperformed than FITI. We have applied the method in the analysis of the UNIX user usage log and mining tandem repeats in protein sequences. There are several directions for future work. The first direction is to develop techniques for mining inter-transaction association rules from a sequence of eventsets, or a transaction database. The second direction is to develop techniques for managing the large number of frequent continuities discovered, by introducing the concept of maximal or closed continuities. More research will be reported in the near future.

Acknowledgements

This work is sponsored by Ministry of Economic Affairs, Taiwan under grant 93-EC-17-A-02-S1-029.

References

1. K.Y. Huang and C.H. Chang. Asynchronous periodic patterns mining in temporal databases. In *Proc. of the IASTED International Conference on Databases and Applications (DBA)*, pages 43–48, 2004.
2. Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovering frequent episodes in sequences. In *Proc. of the First International Conference on Knowledge Discovery and Data Mining*, pages 210–215, 1995.
3. A. K. H. Tung, J. Han H. Lu, and L. Feng. Breaking the barrier of transactions: Mining inter-transaction association rules. In *Proc. of the International Conference on Knowledge Discovery and Data Mining*, pages 297–301, 1999.
4. A. K. H. Tung, J. Han H. Lu, and L. Feng. Efficient mining of intertransaction association rules. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):43–56, 2003.