Mining Closed Episodes from Event Sequences Efficiently

Wenzhi Zhou[1], Hongyan Liu[1], and Hong Cheng[2]

[1] Department of Management Science and Engineering
Tsinghua University, Beijing, China, 100084
{zhouwzh.05,liuhy}@sem.tsinghua.edu.cn
[2] Department of Systems Engineering and Engineering Management,
The Chinese University of Hong Kong, Shatin, N. T., Hong Kong
hcheng@se.cuhk.edu.hk

Abstract. Recent studies have proposed different methods for mining frequent episodes. In this work, we study the problem of mining closed episodes based on minimal occurrences. We study the properties of minimal occurrences and design effective pruning techniques to prune non-closed episodes. An efficient mining algorithm *Clo_episode* is proposed to mine all closed episodes following a breadth-first search order and integrating the pruning techniques. Experimental results demonstrate the efficiency of our mining algorithm and the compactness of the mining result set.

Keywords: Episode, closed episode, frequent pattern, sequence.

1 Introduction

Frequent episode mining in event sequences is an important mining task with broad applications such as alarm sequence analysis in telecommunication networks [1], financial events and stock trend relationship analysis [4], web access pattern analysis and protein family relationship analysis [5]. An event sequence is a long sequence of events. Each event is described by its type and a time of occurrence. A frequent episode is a frequently recurring short subsequence of the event sequence.

Previous studies on frequent itemset mining and sequential pattern mining show that mining closed itemsets [10] or closed sequential patterns [11] is not only an information lossless compression of all frequent patterns, but also an effective way to improve the mining efficiency. A frequent itemset (or sequence) is closed if there is no super itemset (or super sequence) with the same support. To the best of our knowledge, there are some research work on frequent episode or generalized frequent episode mining [2, 3, 7-9], but there is no existing method for mining closed episodes. Therefore, it is natural to raise two questions: (1) what is a closed episode? and (2) how to find closed episodes efficiently?

Following the definitions of closed itemset and closed sequence, we can define closed episode similarly. For example, in Table 1, as event *B* always co-occurs with event *A*, we say event *B* is not closed (the formal definition is given in Section 2)

M.J. Zaki et al. (Eds.): PAKDD 2010, Part I, LNAI 6118, pp. 310–318, 2010.

given a time span constraint of 4 (i.e., the maximum window size threshold) and a frequency constraint of 2 (i.e., the minimum support threshold).

Table 1. An example sequence of events

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Event	*A*	*B*	*C*	*A*	*B*	*D*	*F*	*E*	*F*	*C*	*E*	*D*	*F*	*A*	*B*	*C*	*E*	*A*	*B*

Though the concept of closed episode is similar with that of closed itemset and closed sequence, mining closed episode efficiently is much more challenging. The challenge is caused by two properties in frequent episode mining: (1) there is temporal order among different events in an episode; and (2) we take into account repeated occurrences of an episode in the event sequence.

In this paper, we define a concept of minimal occurrence (a similar concept was defined in [1]) and use minimal occurrence to model the occurrences of an episode in the event sequence. Based on this model, we will develop several pruning techniques to reduce the search space on closed episodes. We propose an efficient algorithm to mine all closed episodes by exploiting the pruning techniques.

The remainder of this paper is organized as follows. We give the problem definition in Section 2. Section 3 presents the novel mining algorithm and our proposed pruning techniques. We demonstrate the performance of our algorithm through extensive experiments in Section 4. Section 5 concludes the paper.

2 Problem Definition

The framework of mining frequent episode was first proposed by Mannila et al. [1]. In [1], two kinds of episodes, serial and parallel episodes, were formally defined. In this paper, we focus on serial episode mining, and we call it episode for simplicity.

Given a set E of event types (or elements), $E=\{e_1, e_2, \ldots, e_m\}$, an event sequence (or event stream) S is an ordered sequence of events, denoted by $S=<(A_1, t_1), (A_2, t_2), \ldots, (A_N, t_N)>$, where $A_i \in E$ is the event that happens at time t_i (we call it the occurrence time of A_i), and $T_s \le t_i \le t_{i+1} < T_e$ for $i=1, 2, \ldots, N-1$. T_s is called the starting time of S, and T_e the ending time. Thus an event sequence can be denoted by (S, T_s, T_e). An **episode** α is an ordered collection of event types, $\alpha=<B_1, B_2, \ldots, B_n>$, where $B_i \in E$ ($i=1, 2, \ldots, n$) and $n<N$. An episode with n events is called an n-length episode, or n-episode. A **window** on an event sequence (S, T_s, T_e) is an event sequence (w, t_s, t_e), where $t_s > T_e$ and $t_e < T_s$. Window w consists of those events (A, t) from S where $t_s \le t < t_e$. The time span $t_e - t_s$ is called the **size** of the window w.

Definition 1 (**Minimal occurrence**). Given an event sequence S and an episode α, a **minimal occurrence** of α is such a time interval $[t_s, t_e)$ that (1) α occurs in $[t_s, t_e)$; (2) there exists no smaller time interval $[t'_s, t'_e) \subset [t_s, t_e)$ that α occurs in it. t_s is the starting time of this occurrence and t_e the ending time; and (3) $t_e - t_s < win$, where win is a user-defined maximum window size threshold, and $t_e - t_s$ is called the **time span** of this minimal occurrence. That is, each occurrence of an episode must be within a window no larger than win.

For an episode α, all of its minimal occurrences are denoted as a set $MO(\alpha)=\{[t_s, t_e) | [t_s, t_e)$ is a minimal occurrence of $\alpha\}$. The **support** of an episode is the number of its minimal occurrences, i.e., $sup(\alpha)=|MO(\alpha)|$. Given a minimum support threshold *minsup*, if $sup(\alpha)\geq minsup$, α is called a frequent episode, or α is frequent.

Definition 2 (**Sub-episode and super episode**)**.** An episode $\beta=<B_1, B_2, \ldots, B_m>$ where $m<n$ is called a **sub-episode** of another episode $\alpha=<A_1, A_2, \ldots, A_n>$, if there are m integers $1\leq i_1<i_2<\ldots<i_m\leq n$ such that $B_1= A_{i1}$, $B_2= A_{i2}$, …, $B_m= A_{im}$. Episode α is called the **super episode** of episode β. If we have $i_1=1$, $i_2=2$, …, $i_m=m$, then α is called the **forward-extension super episode** of β. If we have $i_1=n-m+1$, $i_2 = n-m+2$, …, $i_m=n$, then α is called the **backward-extension super episode** of β. If α is a super episode of β but is neither a forward-extension nor backward-extension super episode of β, α is called the **middle-extension super episode** of β.

Definition 3 (**Closed episode**)**.** Given an event sequence S, an episode α is closed if (1) α is frequent; and (2) there exists no super episode $\beta \supset \alpha$ with the same support as it.

Definition 4 (Forward-closed episode). Episode α is **forward-closed** if (1) α is frequent; and (2) there is no forward-extension super episode of α with the same support as α. Similarly we can define another two kinds of closed episodes.

Definition 5 (Backward-closed episode). Episode α is **backward-closed** if (1) α is frequent; and (2) there is no backward-extension super episode of α with the same support as α.

Definition 6 (Middle-closed episode). Episode α is **middle-closed** if (1) α is frequent; and (2) there is no middle-extension super episode of α with the same support as α.

A closed episode should be forward-closed, backward-closed and middle-closed simultaneously.

Mining Task: Given an event sequence S, a minimum support threshold *minsup*, and a maximum window size threshold *win*, the mining task is to find the complete set of closed episodes from S.

3 Mining Closed Episodes

To enumerate potential closed episodes, we perform a search in a prefix tree as shown in Figure 1. We assume that there is a predefined order, denoted by $\lessdot$, among the set of distinct event types. In our example sequence, we use the lexicographical order, i.e., $A \lessdot B \lessdot C \lessdot D \lessdot E \lessdot F$.

The mining process follows a breadth-first search order. Starting from the root node with the empty episode $\alpha=\varnothing$, we can generate length−1 episodes at level 1 by adding one event to α. Similarly we can grow an episode at level k by adding one event to get length−(k+1) episodes at level k+1. For each candidate episode, we compute its minimal occurrence set from the minimal occurrence sets of its sub-episodes through a "join" operation. Since the episode mining and checking are based on minimal occurrences of episodes, we will first give some properties of minimal occurrences and show how they can be used for pruning search space.

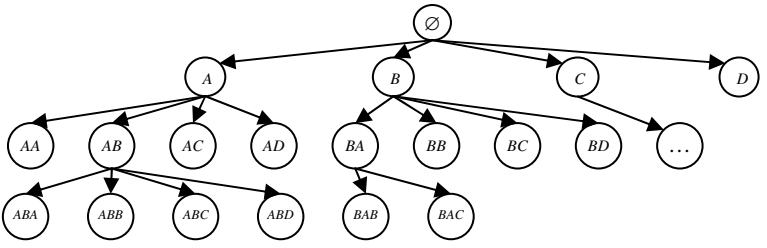

Fig. 1. A tree of episode (sequence) enumeration

Property 1. In an event sequence S, if $[t_s, t_e)$ is a minimal occurrence of episode $\alpha=<A_1, A_2,\ldots, A_n>$, then there must exist two events, (A_1, t_s) and (A_n, t_e-1) in S.

Property 2. Assume $[t_s, t_e)$ and $[u_s, u_e)$ are two minimal occurrences of episode α. If $t_s < u_s$, then we have $t_e < u_e$; if $t_e < u_e$, then we have $t_s < u_s$.

Based on Property 2, minimal occurrences in $MO(\alpha)$ can be sorted in the ascending order of their starting time. Then the order among an episode's minimal occurrences is strict. If one occurrence starts ahead of another, then it must end earlier.

According to Definition 1, the time span of an episode's minimal occurrence is bounded by the window size threshold *win*. Therefore, given a minimal occurrence $[t_s, t_e)$, if $t_e-t_s=win$, the episode cannot be extended forward or backward in the time window $[t_s, t_s+win)$, although some events might be inserted in the middle of it. We define such minimal occurrences as *saturated* minimal occurrences.

Definition 7 (Saturation and expansion). A minimal occurrence $[t_s, t_e)$ of an episode α is **saturated** if the time span $t_e-t_s=win$, the user-specified maximum window size threshold. Otherwise, it is an unsaturated minimal occurrence. An episode α's **saturation,** denoted as α.saturation**,** is defined as the number of saturated minimal occurrences, and its **expansion**, denoted as α.expansion, is the number of unsaturated minimal occurrences.

Apparently, α.saturation+α.expansion=$sup(\alpha)$ holds. For a saturated minimal occurrence of an episode, no additional events can be inserted before the first event or after the last event; otherwise, the maximum window size constraint will be violated. This means, for an episode α we can find its *forward-extension* or *backward-extension* super episode γ only in the unsaturated minimal occurrences. The upper bound of γ's support is the number of α's unsaturated minimal occurrences. Therefore, if γ is frequent, the number of unsaturated minimal occurrences in $MO(\alpha)$ must be no smaller than *minsup*. If one episode has less than *minsup* unsaturated minimal occurrences, then none of its *forward-extension* or *backward-extension* super episodes is frequent. In this case, it is unnecessary to generate and check its super episodes. Therefore, we have following two lemmas.

Lemma 1: If the expansion of an episode α is smaller than the minimum frequency threshold *minsup*, i.e., $\alpha.expansion<minsup$, then all forward-extension and backward-extension super episodes of α are infrequent.

Lemma 2: Given a frequent episode α and its minimal occurrence set $MO(\alpha)$, if (1) $\alpha.saturation>0$, and (2) there is a saturated minimal occurrence $\omega \in MO(\alpha)$ that cannot be extended in the middle, then α must be closed.

The algorithm, *Clo_episode*, is developed to find all closed episodes in an event sequence. Its major steps are given in Figure 2.

The algorithm structure is as follows. It first generates the set of length-1 frequent episodes F_1 (lines 1-4). The event sequence S is scanned and all distinct single events are generated as length−1 episodes with their minimal occurrences. Then, *Clo_episode* generates closed episodes level by level in an iterative way through the *while* loop (lines 5-19).

Each iteration in the while loop generates the set of candidate episodes C_k and the set of frequent episodes F_k. This process iterates until F_k is empty. For a non-empty set F_k, it produces F_{k+1} which is the frequent episode set of length−(k+1) and CF_k which is the set of length−k closed episodes. Episodes in F_k are processed in a predefined order as we explained above.

Algorithm *Clo_episode*

Input: Event sequences S, maximum window size threshold *win*, minimum support threshold *minsup*;
Output: the set of closed episodes CF

Method:

1. C_1= the set of length−1 episodes in S
2. scan S and compute $MO(\alpha)$ for all $\alpha \in C_1$;
3. k=1;
4. F_1={$\alpha \in C_1$ | α is frequent}
5. **while** F_k != Φ do
6. **for** each episode $\alpha \in F_k$ such that $\alpha.forward$!=0 do
7. C_{k+1}=*Gen_candiate*(α, F_k);
8. C_{k+1}=*Prune*(C_{k+1});
9. C_{k+1}=*Gen_MO*(C_{k+1});
10. **for** each candidate episode $\gamma \in C_{k+1}$ do
11. **if** $sup(\gamma) \geq minsup$, put γ into F_{k+1}
12. **if** there is length−k subepisode β of γ such that $\beta.closed$=-1 and $sup(\gamma)=sup(\beta)$ do
13. set $\beta.closed$=0
14. **if** γ is a middle-extension super episode of β, call *Clo_prune*(α, β, γ);
15. **end for**
16. **end for**
17. CF_k={$\alpha \in F_k$ |$\alpha.closed$!= 0}
18. k=k+1;
19. **end while**
20. CF={α |$\alpha \in CF_i$ ($1 \leq i < k$) and there is no super episode β of α such that $sup(\beta)=sup(\alpha)$};

Output CF

Fig. 2. Major steps of Algorithm *Clo_episode*

For each episode $\alpha \in F_k$ which can be extended forward (line 6), function *Gen_candiate*(α, F_k) is called to generate α's candidate super episodes of length−(k+1) (line 7). A super episode is generated by combining α and another length−k episode in F_k that shares the first (k−1) events with α. Then function *Prune*(C_{k+1}) is called to prune those candidates that cannot be frequent (line 8). Next *Gen_MO*(C_{k+1}) is invoked to generate the minimal occurrences for each candidate episode. It also computes saturation and expansion of

each candidate episode and checks whether they can be extended forward or backward and whether they are closed. Due to space limitation, we omit the details of the two functions. Based on the computed information, each frequent episode γ is added into F_{k+1} (line 11); the sub-episodes of γ that have the same support as γ will be marked not closed (lines 12-13). For those sub-episodes β of γ that share the first and last events with γ, function *Clo_prune*(α, β, γ) is invoked to see if it can be pruned (line 14). The details of function *Clo_prune* are given in Figure 3.

```
Clo_prune(α, β, γ)
1 Sort minimal occurrences in MO(β) and MO(γ) respectively by starting time ascendingly
2 Initialize flag=true
3 for i=1 to |MO(β)| do
4   if MO(γ)[i].t_s!=MO(β)[i].t_s or MO(γ)[i].t_e!=MO(β)[i].t_e do
5     flag=false
6     break
7 end for
8 if flag=true do
9   Remove β from F_k;
10  if (β≺α) remove all β's forward-extension super episodes from F_{k+1};
11 return
```

Fig. 3. Major steps of Function *Clo_prune*

Function *Clo_prune* compares the minimal occurrences of episode β and that of its middle-extension super episode γ. If $MO(\gamma)=MO(\beta)$, then β is not closed. All of β's forward-extension super episodes are not closed either. Due to space limitation, we omit the detail of proof.

4 Experiments

In this section we will present experimental results on synthetic datasets. We also conducted experiments on a real dataset. But due to space limitation, we will only report the results of synthetic datasets. We evaluate the efficiency of our proposed algorithm as well as the reduction in the number of episodes generated, comparing with *MINEPI* proposed by Mannila et al. [1] for frequent episode mining. To test the effectiveness of the pruning techniques for pruning non-closed episodes, we disable the function *Clo_prune* and get another algorithm called *Clo_episode_NP* without the pruning techniques. All of our experiments were performed on a PC with 3Ghz CPU, 2GB RAM, and 300GB hard disk running windows XP.

We design a synthetic dataset generator, by which we can evaluate the performance of different algorithms with different data characteristics. The generator takes five parameters. The parameter *NumOfPatterns* is the number of potentially closed episodes in the final sequence, *MaxLength* and *MinLength* are the maximum and minimum length of potential episodes respectively, *NumOfWindows* is used to control the length of the whole sequence, and *win* is the size of a window. Due to space limitation, we omit the detail of the generator.

To generate event sequences, we fix the value of *MinLength*, and vary the other five parameters: *NumOfPatterns*(*P*), *MaxLength*(*L*), *NumOfWindow*s(*N*), *win*(*W*) and *minsup*. Thus we create five groups of datasets, each of which is generated by varying one parameter and fixing the other four parameters as shown in Table 2.

Table 2. Five groups of synthetic datasets

Group	Datasets	*P*	*L*	*N*	*W*	*Minsup*
1	PxL12N4000W14	100-200	12	4000	14	15
2	P200L12NxW14	200	12	3200-4800	14	20
3	P200LxN4000W16	200	9-14	4000	16	15
4	P200L12N4000Wx	200	12	4000	10-16	20
5	P200L12N4000W14	200	12	4000	14	10-35

For each group of datasets, we plot the running time and the number of frequent/closed episodes in two figures. The results are shown in Figures 4 to 13. The *y*-axes are in logarithmic scales in Figures 5, 7, 9, 11 and 13.

From Figure 4 we can see that *Clo_episode* significantly outperforms *MINEPI* and *Clo_episode_NP*. This shows that the pruning techniques in *Clo_episode* are very effective. In addition, the running time of *Clo_episode* is not affected much as *P* increases, but that of the other two algorithms increases dramatically. Figure 5 shows that the number of closed episodes found by *Clo_episode* is much smaller than the number of frequent episodes output by *MINEPI*.

Figures 6 and 7 show a similar result. As *N* increases, the sequence becomes longer. For a fixed *minsup* threshold, the number of frequent episodes by *MINEPI* increases a lot, but the number of closed episodes does not increase as much.

Figures 8 and 9 show that the number of frequent episodes output by *MINEPI* increases significantly with *L*. Accordingly the running time increases. Figure 8 shows that *Clo_episode* is much more efficient than *MINEPI* and *Clo_episode_NP*.

Figures 10 and 11 show that when *win* increases from 10 to 14, the running time becomes longer, the number of frequent episodes becomes larger, but the number of closed episodes does not increase much. When *win* is greater than 14, the running time of all three algorithms does not increase. This is because the average length of potential closed episodes is fixed at 12 in the sequence.

Figures 12 and 13 show that, as *minsup* increases, the number of frequent and closed episodes decreases. The running time decreases accordingly.

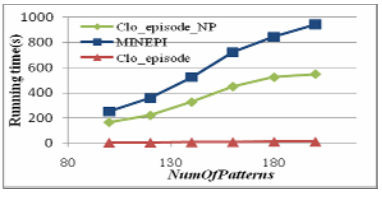

Fig. 4. Running time vs *P*

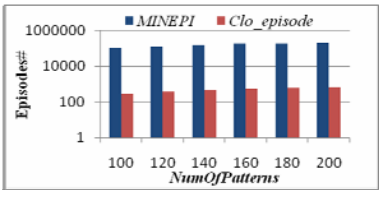

Fig. 5. No. frequent/closed episodes vs *P*

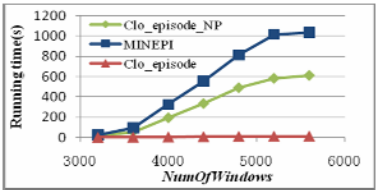

Fig. 6. Running time vs *N*

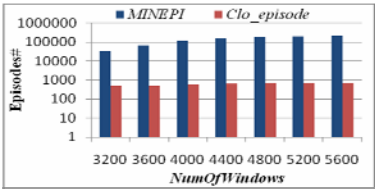

Fig. 7. No. frequent/closed episodes vs *N*

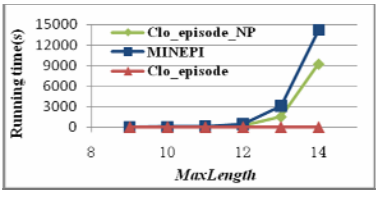

Fig. 8. Running time vs *L*

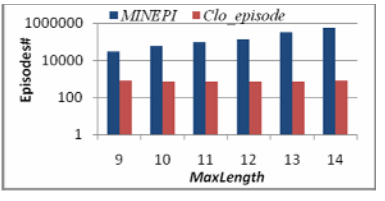

Fig. 9. No. frequent/closed episodes vs *L*

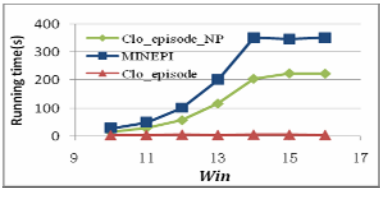

Fig. 10. Running time vs *W*

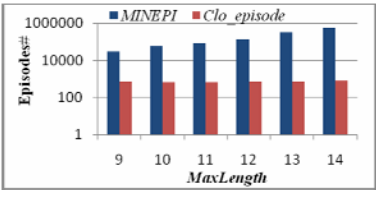

Fig. 11. No. frequent/closed episodes vs *W*

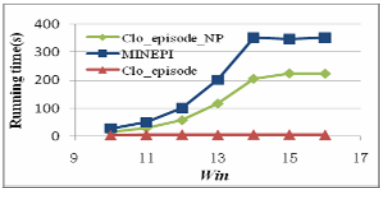

Fig. 12. Running time vs *minsup*

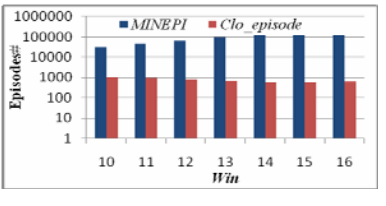

Fig.13. No. frequent/closed episodes vs *minsup*

Overall, from these five groups of experiments we can see that our algorithm *Clo_episode* performs much better than *MINEPI* and *Clo_episode_NP*.

5 Conclusion

Frequent episode mining is an important mining task in data mining. In this paper, we study how to mine closed episodes efficiently. We proposed a novel algorithm

Clo_episode, which incorporates several effective pruning strategies and a minimal occurrence-based support counting method. Experiments demonstrate the effectiveness and efficiency of these methods.

Acknowledgments. This work was supported in part by the National Natural Science Foundation of China under Grant No. 70871068, 70621061 and 70890083.

References

[1] Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of Frequent Episodes in Event Sequences. Data Mining and Knowledge Discovery 1(3), 259–289 (1997)
[2] Mannila, H., Toivonen, H.: Discovering generalized episodes using minimal occurrences. In: KDD 1996, Portland, OR (1996)
[3] Laxman, S., Sastry, P.S., Unnikrishnan, K.P.: A Fast Algorithm For Finding Frequent Episodes In Event Streams. In: KDD 2007, SanJose, CA (2007)
[4] Ng, A., Fu, A.: Mining Frequent Episodes for Relating Financial Events and Stock Trend. In: PAKDD 2003, Seoul, Korea (2003)
[5] Baixeries, J., Casas-Garriga, G., Balcázar, J.L.: Mining unbounded episodes from sequential data
[6] Meger, N., Rigotti, C.: Constraint-based mining of episode rules and optimal window sizes. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) PKDD 2004. LNCS (LNAI), vol. 3202, pp. 313–324. Springer, Heidelberg (2004)
[7] Gwadera, R., Atallah, M.J., Szpankowski, W.: Reliable detection of episodes in event sequences. In: ICDM 2003, Melbourne, FL (2003)
[8] Gwadera, R., Atallah, M.J., Szpankowski, W.: Markov models for identification of significant episodes. In: SDM 2005, Newport Beach, CA (2005)
[9] Laxman, S., Sastry, P.S., Unnikrishnan, K.P.: Discovering frequent episodes and learning Hidden Markov Models: A formal connection. IEEE TKDE 17(11), 1505–1517 (2005)
[10] Wang, J., Han, J., Pei, J.: CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In: KDD 2003, Washington, DC (2003)
[11] Yan, X., Han, J., Afshar, R.: CloSpan: Mining Closed Sequential Patterns in Large Databases. In: SDM 2003, San Francisco, CA (2003)