Continuously Matching Episode Rules for Predicting Future Events over Event Streams

Chung-Wen Cho[1], Ying Zheng[2], and Arbee L. P. Chen[3]

[1] Department of Computer Science, National Tsing Hua University, Taiwan, R.O.C.
[2] Department of Computer Science, Fudan University, China
[3] Department of Computer Science, National Chengchi University, Taiwan, R.O.C.
alpchen@cs.nccu.edu.tw

Abstract. Predicting future events has great importance in many applications. Generally, rules with *predicate events* and *consequent events* are mined out, and then current events are matched with the predicate ones to predict the occurrence of consequent events. Many previous works focus on the rule mining problem; however, little emphasis has been attached to the problem of predicate events matching. As events often arrive in a stream, how to design an efficient and effective event predictor becomes challenging. In this paper, we give a clear definition of this problem and propose our own method. We develop an event filter and incrementally maintain parts of the matching results. By running a series of experiments, we show that our method is efficient and effective in the stream environment.

Keywords: Continuous query, episode, event stream, prediction.

1 Introduction

In many applications, *events* such as specific TCP connections in an intrusion detection system [10] are recorded for the predicting of future events. Generally speaking, there are two steps for the event prediction problem. The first step is to derive event associations represented as rules from the past events. The second one is to use the discovered rules to predict future events when given a recent record of events. We now explain these two steps by an example, and show the motivation of our work.

Fig.1 shows an example of the discovered rule in the form $\alpha \Rightarrow \beta$ where α is called the *predicate* and β the *consequent*. α and β are both represented by a directed acyclic graph, where each vertex represents an event, and each edge from vertex v to vertex u indicates that the event corresponding to vertex v should occur before that corresponding to vertex u. To be specific, according to the predicate α in Fig.1, event a should precede events b and c, and event b should precede event d. Additionally, there are two time bounds associated with the rule and the predicate, respectively. For example, in Fig.1, if all the events occur within the time bound of 7 time units in accord with the specified temporal orders in the predicate, we can predict that all the events in the rule will, with a certain probability, appear within 11 timestamps

according to their temporal orders indicated in the rule. The first step of the event prediction problem mines out such kind of rules with two time bounds (called *episode rule*) from the past events.

In the second step, whether all the events in the predicate have appeared according to their specified partial orders within the time bound (denoted as *rule matching* problem) is determined to predict the happening of the events in the corresponding consequents. For example, suppose the events coming from timestamp 1 to 9 are as depicted in Fig. 2, and we are to match the episode rule in Fig. 1. Notice that events a, b, c, and d occur within the time interval [3, 8), which satisfies the temporal constraint in the predicate. Thus, we should give the alarm that the consequent, or event f, may come within the time interval [8, 14) with a certain probability. We denote the occurrences of the events corresponding to the matching of the predicate as *predicate episode occurrence*.

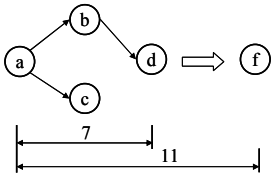

Fig. 1. An episode rule

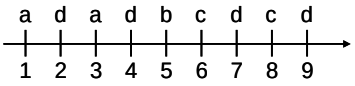

Fig. 2. A stream of events

The discovery of episode rules has been widely discussed over the past few years [6], [7]. However, little attention has been given to solve the important phase of episode rule matching. Since events arrive in streaming in all the applications mentioned above, how to efficiently match a number of episode rules in this environment becomes an important and difficult task. In this paper, we aim at this problem of continuously matching episode rules over the stream of events.

The main challenges of this problem are stated below. 1) Many predicate episode occurrences of a rule can exist simultaneously by sharing the same occurrences of events over the stream. However, only the occurrences that give non-repetitive predictions of the occurrences of the consequents are what we concern about. Take the example of the episode rule in Fig. 1 and the stream of events in Fig. 2. It can be seen that from the two predicate episode occurrences, {(a,1), (b,5), (c,6), (d,7)} and {(a,3), (b,5), (c,6), (d,7)} (we use (e,t) to denote the event e with an occurring time t), we predict [8,12) and [8, 14) as the occurring time interval of event f, respectively. Since the predicted interval [8,12) is included in [8,14) and becomes trivially redundant, the occurrence {(a,1), (b,5), (c,6), (d,7)} can be ignored. 2) The structure of the episodes can be complex. High precision should be emphasized to effectively deal with all the possible combinations of events within the specified time bounds to match the episodes. 3) There are a large number of predicate episodes in different rules to be matched simultaneously. Moreover, events usually come in bursts. There is only a limited time to make matches for all the rules. A prompt episode detector is hence required.

Our problem is related to three research topics. 1) Mining graph patterns from event or graph data sets [5], [6], [11]. The goals of these papers are essentially different from ours since we aim at continuous queries and they target on mining

process. 2) Efficient graph indexing for pattern searching [3], [4]. Nevertheless, all these methods are applied to a static graph database searching, which is very different from our work of continuous retrieval in the streaming environment. 3) Graph filtering in the stream environment [8], [9] and the query of the temporal relations over DBMSs [2]. These works are similar to ours. However, we are to retrieve episodes within the specified time bounds and the repetitive reports of predicted time intervals should be avoided. So we cannot directly apply these algorithms to address our problem.

In this paper, we give a clear definition to the rule matching problem for event prediction, where the concepts of *minimal episode occurrence*, *latest episode occurrence* and *rejected event occurrence* were introduced to address the first challenge mentioned above. With the constraints in our problem definition, the retrieval of only user-required episode occurrences is assured. We then propose the method *ToFel* to solve this problem. ToFel makes use of the topological characteristic of the predicate episode, and develops its own pruning criteria. More specifically, ToFel finds the predicate episode occurrences by incrementally maintaining parts of the user-required episode occurrences, and thus avoid the backward scan of the stream. It constructs one event filter for each predicate episode to be matched. The filters continuously monitor the newly arrived events and only keep those which are likely to be parts of the predicate episode occurrences. By running a series of experiments with respect to different scales and distributions of the query set and the stream, we show that ToFel is efficient and effective in the stream environment.

The remainder of the paper is organized as follows. Section 2 gives a detailed description of the problem statement. Section 3 presents our rule matching algorithm. The experimental results are discussed in Section 4. We give the conclusion and future directions of our work in Section 5.

2. Problem Statement

The episode is a widely used representation for the associations of events. In this section, we first give the definitions related to the episode, and then present the basic concepts concerning the rule matching problem.

Episode: An *episode* is a directed acyclic graph g, where each vertex corresponds to an event, and each directed edge (u,v) indicates that the event corresponding to u must precede that corresponding to v. We call this precedence a temporal and transitive order $\prec$ between vertex u and vertex v. Denote *V(g)* as a vertex set, *E(g)* as an edge set, and $\varepsilon(v)$ as the event corresponding to vertex v. The *sink* of the graph g is defined as the vertex with out-edge degree being equal to 0. For convenience, we focus on episodes consisting of vertices corresponding to different events. However, our techniques can be extended to episodes containing two or more vertices corresponding to an identical event.

Episode occurrence: An *event stream* can be represented as $\hat{S} = <(a_1,t_1), (a_2,t_2), \ldots (a_n,t_n) \ldots>$, where (a_i,t_i) represents that event a_i occurs at time t_i , i = 1, 2, …, n, …, and $t_1<t_2< \ldots<t_n$ …. Given an *event sequence* $S = <(a'_1,t'_1), (a'_2,t'_2) \ldots (a'_m,t'_m)>$,

where $t'_1<t'_2< \ldots <t'_m$, we define the *time interval* or *interval* of S as $[t'_1,t'_m+1)$, and the *start time* and *end time* of S as t'_1 and t'_m+1, respectively. Given an episode α with its time bound ω_α, the *episode occurrence* or *occurrence* of α over $\hat{S}$ is an event sequence S with time interval $[t_s,t_e)$ satisfying that: 1) There exist m integers i_1, i_2,…, i_m, such that $1\le i_1<i_2< \ldots <i_m$ and $\forall 1\le j\le m$: $a'_j=a_{i_j}$ and $t'_j=t_{ij}$; 2) Events corresponding to vertices in α have one-to-one mappings to events in S; 3) The occurrence orders of events in S are consistent with the $\prec$ constraints in α; and 4) $t_e - t_s \le \omega_\alpha$. We denote the *mapping occurrence* of the vertex v in α as the occurrence of the event corresponding to v in S.

Episode Rule: An *episode rule* R is a 5-tuple (α, β, ω_α, $\omega_{\alpha\beta}$, conf). Here, α and β are episodes representing the predicate and consequent of R, respectively. ω_α and $\omega_{\alpha\beta}$ ($\omega_\alpha<\omega_{\alpha\beta}$) correspond to the time bounds of α and $\alpha\beta$ ($\alpha\beta$ is an episode satisfying that each vertex in $\alpha \prec$ any vertex in β), respectively. The interpretation of R is that if α has an occurrence O with interval $[t_s,t_e)$, β will occur during interval $[t_e,t_s+\omega_{\alpha\beta})$ with probability conf. We denote $[t_e,t_s+\omega_{\alpha\beta})$ as the *predicted interval* of the occurrence O.

Given a set of episode rules, our problem is to continuously retrieve the episode occurrence of the predicate α with the time bound ω_α over the event stream, and give non-repetitive information of the predicted intervals for the consequent β. We now introduce the concepts of mi-latest occurrence and rejected event occurrence, and give a clear definition of the rule matching problem.

Definition 1. *Minimal occurrence.* A minimal occurrence O of a predicate episode α is an occurrence with predicted interval $[t_{s1},t_{e1})$ satisfying that there does not exist any other occurrence of α with predicted interval $[t_{s2},t_{e2})$, s.t. $t_{s1}\le t_{s2}<t_{e2}\le t_{e1}$, and $t_{e2}-t_{s2}<t_{e1}-t_{s1}$.

We can focus only on the minimal occurrences and report their predicted intervals without loss of any desired information of the prediction. Meanwhile, there may exist several different occurrences with the same time interval, which give the same predicted interval for the consequent. In the following, we introduce the concept of latest occurrences as a representative of the occurrences with the same interval, and match only the latest occurrences so as to avoid the repetitive reports of the same predicted interval.

Definition 2. *Latest occurrence of an event.* Let $\{(e,t_1), (e,t_2), \ldots (e,t_k)\}$ be the set of occurrences of event e in time interval $[t_s,t_e)$ on the event stream □, where $t_s\le t_1<t_2< \ldots <t_k<t_e$. We call (e,t_k) the *latest occurrence of e* in $[t_s,t_e)$ on □.

Definition 3. *Latest occurrence of an episode*. Given a predicate episode α with vertices v_1, v_2, …, v_n and its occurrence $O_r = <(\varepsilon(v_1),t_1), (\varepsilon(v_2),t_2), \ldots (\varepsilon(v_n),t_n)>$ in time interval $[t_s,t_e)$ on the event stream $\hat{S}$. O_r is called the *latest occurrence* of α in $[t_s,t_e)$ if both of the following conditions hold: 1) Let v_{j1}, v_{j2}, … v_{jx} be the sink vertices of α. $(\varepsilon(v_y),t_y)$ is the latest occurrence of $\varepsilon(v_y)$ in time interval $[t_s,t_e)$, $y=j_1, \ldots, j_x$; 2) for a non-sink vertex v_k of α, let v_{k1}, v_{k2}, … v_{km} be the children of v_k, where $1\le k<n$, and $t_s\le t_k<t_{k1}<t_{k2}< \ldots <t_{km}<t_e$. $(\varepsilon(v_k),t_k)$ is the latest occurrence of $\varepsilon(v_k)$ in time interval $[t_s, t_{k1})$.

Property 1. Let O be a latest occurrence of a predicate episode α with time interval $[t_s,t_e)$. If there exist minimal occurrences of α with the end time equal to t_e, O is one of the minimal occurrences of α.

For example, consider Fig. 1 and Fig. 2, where the latest occurrences of events a, b, c, and d in the interval [1,8) are (a,3), (b,5), (c,6), and (d,7), respectively. The latest occurrence of the predicate episode in the interval [1,8) is the occurrence O = <(a,3), (b,5), (c,6), (d,7)>, which is also a minimal occurrence.

Definition 4. *Mi-latest occurrence.* The *mi-latest occurrence* of a predicate episode α is defined as the occurrence which is both a minimal occurrence of α and a latest occurrence of α.

We define the rule matching problem by the concept of mi-latest occurrence. The rule matching problem is to give predicted intervals of only the mi-latest occurrences to the user.

Definition 5. *The rejected event occurrence*. Given a predicate episode α with vertices v_1, v_2, … v_n and its mi-latest occurrence O = <($\varepsilon(v_1)$,t_1), ($\varepsilon(v_2)$,t_2), … ($\varepsilon(v_n)$,t_n)> on the event stream □. The *rejected event occurrence* deduced from O is defined recursively as following: 1) ($\varepsilon(v_1)$,t_1) is a rejected event occurrence; 2) let v_{i1}, v_{i2}, … v_{im} be the children of v_i, $1< i, i_1, i_2, \ldots, i_m \le n$. If ($\varepsilon(v_i)$,$t_i$) is a rejected event occurrence, ($\varepsilon(v_{ij})$,t_{ij}) is a rejected event, $\forall j=i_1, i_2, \ldots, i_m$, if there is no occurrence of $\varepsilon(v_i)$ in interval (t_i ,t_{ij}).

The essential of the rejected event occurrence deduced from the mi-latest occurrence O is that it can not be part of any other mi-latest occurrence that will appear later than O.

Lemma 1. Given a latest occurrence O = <(e_1,t_1), (e_2,t_2), … (e_n,t_n)> of a predicate episode α. If (e_i,t_i) is not a rejected event occurrence, $\forall 1\le i\le n$, O is a minimal occurrence of α (For the detailed proof of the lemmas in this paper, please refer to our technical report [1]).

To conclude this section, only the latest occurrences with no rejected event occurrences are the mi-latest occurrences we are looking for. This is the basic of our approach, whose correctness can be guaranteed if such occurrences are always targeted during the rule matching process.

3. The Proposed Approach: ToFel

In the following, we present ToFel for the match of a given episode rule R = (α, β, ω_α, $\omega_{\alpha\beta}$, conf). ToFel builds a queue of event occurrences that are likely to be parts of mi-latest occurrences for α and maintains the queue at each timestamp. We first discuss which event occurrences should be kept in the queues and on which condition we should remove the stored occurrences from the queues in the process of continuously monitoring of the stream.

For each vertex of α, we implement a queue to store its corresponding event occurrences. Let Q_v be the queue for vertex $v\in V(\alpha)$. Intuitively, for any event $\varepsilon(v)$

arrives at time t, we should keep this event occurrence for contributing a mi-latest occurrence of α with another coming event occurrence.

As time passes and more and more events arrive, we maintain the queues and only keep those useful occurrences. Since the queues is to store only the occurrences likely to contribute to the results, the occurrences whose occurring time t' satisfies that $t' + \omega_\alpha \geq t$ (the current time) should be removed. In this condition, the maintenance of the queue is invoked. We call this kind of invocation of queue maintenance *time-out invocation*. Besides, as suggested in Definition 5 and Lemma 1, once we find out a mi-latest occurrence, we should adjust the queue by removing the rejected event occurrences. This condition is called *rejected-event invocation.* Both invocation forms are important for the correctness of our answer as well as the space saving.

Definition 6. *The nearest parent occurrence*. Given any two event occurrences $(\varepsilon(v),t)$ and $(\varepsilon(u),t')$, where $u, v \in V(\alpha)$. If v is a parent of u and $t<t'$, we call $(\varepsilon(v),t)$ a *parent occurrence* of $(\varepsilon(u),t')$, and $(\varepsilon(u),t')$ a *child occurrence* of $(\varepsilon(v),t)$. Moreover, if $(\varepsilon(v),t)$ is the latest occurrence of $\varepsilon(v)$ in interval $[t_s,t')$, we call $(\varepsilon(v),t)$ a *nearest parent occurrence* of $(\varepsilon(u),t')$.

To easily maintain the queues, we establish a link called *nearest link*, or *nlink* to link from child occurrence to its nearest parent ones, and maintain a counter for each parent occurrence indicating the number of nlinks between it and its child occurrences.

In the following, we point out the principles for the maintenance of the queues and the further removal of other useless occurrences. A first, if there is no nlink between $(\varepsilon(v),t')$ and any of v's child occurrences when $(\varepsilon(v),t)$ arrives, it indicates that no child occurrence of $(\varepsilon(v),t')$ occurs between time t' and t. Therefore, when a child occurrence of $(\varepsilon(v),t')$ comes, its occurring time must be larger than t, and it will select $(\varepsilon(v),t)$ as its nearest occurrence rather than $(\varepsilon(v),t')$. Therefore, the occurrence $(\varepsilon(v),t')$ can not be linked by any of the later child occurrences, and can be removed without loss of any desired result. Moreover, if an occurrence is removed by above principle, the nlinks between it and its nearest parent occurrences are broken. The counts of those parent occurrences should be decreased by one accordingly. This may further cause the zero count of some of its parent occurrences. We should then adopt about principle again, and remove the parent occurrences with zero counts, except for the tail occurrences. Hence, the above procedure may be propagated from a child vertex to several of its ancestors. We call this queue maintenance principle the *parent-occurrence maintenance*. Moreover, when an occurrence $(\varepsilon(v),t)$ is removed, the event occurrences that can be reached from $(\varepsilon(v),t)$ by nlinks should be removed too. This maintenance principle is called *child-occurrence maintenance*.

To sum up, once an event $\varepsilon(v)$ comes at time t, $v \in V(\alpha)$, if v is not a sink of α, we normally push $(\varepsilon(v),t)$ into Q_v, and perform the parent-occurrence maintenance. Otherwise, if v is a sink of α, we detect whether the mi-latest occurrence of α exists. This includes three processes: 1) The time-out invocation calls the queue maintenance, 2) push $(\varepsilon(v),t)$ into Q_v, and perform the parent-occurrence maintenance, and 3) check whether there exists a mi-latest occurrence according to Lemma 2 presented later. If a mi-latest occurrence exists, the rejected-event invocation will call the queue maintenance procedure. In the following, we show how to retrieve the mi-latest occurrence from the queues of event occurrences.

Property 2. Let v be a sink vertex of episode α. There is at most an occurrence $(\varepsilon(v),t)$ kept in Q_v, and $(\varepsilon(v),t)$ is the latest occurrence of $\varepsilon(v)$ so far.

Lemma 2. Let $v_1, v_2, \ldots v_n$ be the vertices of episode α and $v_i, v_{i+1}, \ldots v_n$ be the sinks of α. If there is a mapping occurrence of v_j kept in Q_{vj}, $\forall i \le j \le n$, there must exist a mi-latest occurrence O of α with interval $[t_1,t_n+1)$, and O must be $<(\varepsilon(v_1),t_1), (\varepsilon(v_2),t_2), \ldots (\varepsilon(v_n),t_n)>$, where $t_n+1-t_1 \le \omega_o$, and $(\varepsilon(v_k),t_k)$ is the 1st element in $Q_{vk}, \forall 1 \le k \le n$.

The correctness of ToFel can be proved as follows. Whenever a new mi-latest occurrence exists, its last element must correspond to a sink of α. Therefore, when each sink occurrence comes, we check if there exists a mi-latest occurrence by Lemma 2. And, we can prove that the time complexity of ToFel is O(n). [1]

4. Experimental Results

In this section, we evaluate the performances of DirectMatch [1] and ToFel by a series of experiments on synthetic data. The data is generated by the synthetic data generator [1]. We set various parameters to evaluate our method in the running time as well as the scalability on the structure of the episode and the size of the dataset. For the parameter settings, please refer to [1].

Fig. 3 shows the average execution time at each timestamp with respect to the number of episode rules to be matched. Though the time increases with the query number, ToFel always outperforms DirectMatch. And the increasing ratio in running time of ToFel is less than that of DirectMatch significantly. This can be explained that when matching an episode, ToFel only concerns the events likely to form the mi-latest occurrences, while DirectMatch usually repeatedly retrieves the kept events many of which are not even relevant to the episode. We also show the performance with respect to the number of vertices in the episode as shown in Fig. 4. And the result shows the slow increase in CPU time as well as the smaller time requirement of ToFel compared with DirectMatch. Finally, we compare the two approaches in their scalability with respect to the size of the event stream. As shown in Fig. 5, both approaches have the constant average running time at each timestamp no matter how the size of the stream changes.

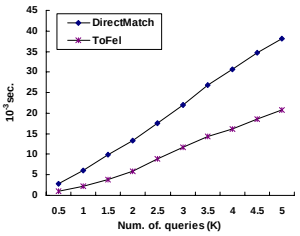

Fig. 3. Running time for different query numbers

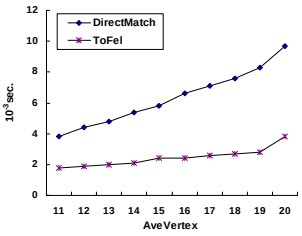

Fig. 4. Running time for different AveVertex values

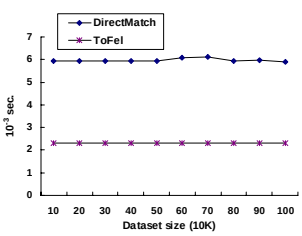

Fig. 5. Running time for different dataset sizes

5. Conclusion and Future Work

In this paper, we propose a novel, deterministic and efficient approach to continuously match the episode rules over event streams for the predicting of future events. We introduce the concepts of mi-latest occurrence and rejected event occurrences such that no repetitive predicted intervals are reported. Besides, we build and continuously maintain the queue of events which are likely to contribute to the desired occurrences in an efficient time once a new event arrives. This leads to a prompt reaction towards the desired report of episode occurrences that may burst at one timestamp. Moreover, a series of experiments demonstrate the high performance of our approach in real processing time as well as the stability with respect to the number of queries, the number of vertices, and the size of event streams. For the future work, we will focus on utilizing the common substructures among the predicate episodes so as to more efficiently process a batch of them simultaneously.

Acknowledgments This work was partially supported by the NSC Program for Advanced Technologies and Applications for Next Generation Information Networks (II) under the grant number NSC 95-2752-E-007-004-PAE, and the NSC under the contract number 95-2627-E-004-002-.

References

1. Cho, C.W., Y. Zheng, and A.L.P. Chen. Continuously Matching Episode Rules for Predicting Future Events over Event Streams. Tech. Report CS-1006-05, Department of Computer Science, National Tsing Hua University, October 2006.
2. Chomicki J. History-less Checking of Dynamic Integrity Constraints. In Proceedings of the 8th International Conference on Data Engineering, 1992, 557-564.
3. Giugno, R. and D. Shasha. GraphGrep: A Fast and Universal Method for Querying Graphs. In 16th International Conference on Pattern Recognition, 2002, 112-115.
4. He, H. and A.K. Singh. Closure-Tree: An Index Structure for Graph Queries. In Proceedings of the 22nd International Conference on Data Engineering, 2006, p. 38.
5. Hsieh, C.E., Y.H. Wu and A.L.P. Chen. Discovering Frequent Tree Patterns over Data Streams. In Proceedings of the 6th SIAM International Conference on Data Mining, 2006.
6. Mannila, H., H. Toivonen, and A.I. Verkamo. Discovering Frequent Episodes in Sequences. In Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining, 1995, 210-215.
7. Mannila, H., H. Toivonen, and A.I. Verkamo. Discovery of Frequent Episodes in Event Sequences. Data Mining and Knowledge Discovery, 1(3), 1997, 259-289.
8. Olteanu, D., T. Kiesling, and F. Bry. An Evaluation of Regular Path Expressions with Qualifiers against XML Streams. In Proceedings of the 19th International Conference on Data Engineering, 2003, 702-704.
9. Peng, F. and S.S. Chawathe. XPath Queries on Streaming Data. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2003, 431-442.
10. Qin, M. and K. Hwang. Frequent Episode Rules for Internet Anomaly Detection. IEEE International Symposium on Network Computing and Applications, 2004, 161-168.
11. Yan, X. and J. Han. gSpan: Graph-Based Substructure Pattern Mining. In Proceedings of the 2002 IEEE International Conference on Data Mining, 2002, 721-724.