

# Finding Event-Oriented Patterns in Long Temporal Sequences

Xingzhi Sun, Maria E Orlowska, and Xiaofang Zhou

The University of Queensland, Australia  
{sun, maria, zxf}@itee.uq.edu.au

**Abstract.** A major task of traditional temporal event sequence mining is to find all frequent event patterns from a long temporal sequence. In many real applications, however, events are often grouped into different types, and not all types are of equal importance. In this paper, we consider the problem of efficient mining of temporal event sequences which lead to an instance of a specific type of event. Temporal constraints are used to ensure sensibility of the mining results. We will first generalise and formalise the problem of event-oriented temporal sequence data mining. After discussing some unique issues in this new problem, we give a set of criteria, which are adapted from traditional data mining techniques, to measure the quality of patterns to be discovered. Finally we present an algorithm to discover potentially interesting patterns.

## 1 Introduction

Finding frequent patterns in a long temporal event sequence is a major task of temporal data mining with many applications. Substantial work [1-5] has been done in this area, however, it often focuses on specific application requirements.

In some domains, there is a need to investigate the dependencies among selected types of events. A temporal event sequence can then be formed by integrating data sets from different domains according to common features such as the time order of each event. Naturally, in such a sequence, not every type of event may have equal impact on the overall data analysis goals. We may be particularly interested in a specific event type and any event patterns related to the occurrences of this event type. The temporal relationship between event patterns and specific type of events can be specified by temporal constraints. The temporal constraints, such as the size of time window considerations, are important in the knowledge discovery from the temporal sequence, because they closely relate to the sensibility of mining results.

An application of such kind of problem could be earthquake prediction. Many types of events, which may relate to earthquakes, are recorded by independent devices but they have one common reference point - the recorded time of event occurrence. Consequently, a temporal sequence is formed by ordering timestamps of these events. Assuming that our goal relates to ability to set up some proactive measures in an earthquake situation, the evidence of the actual earthquake is the most important event type in the sequence. We would like to identify all patterns

of different types of recorded events (excluding or including the earthquake type itself) which potentially indicate the possibility of an earthquake situation. The temporal constraint would be then the length of the period of interest preceding the actual earthquake occurrence.

Another application is telecommunication network fault analysis. A telecommunication network can be viewed as a hierarchical network, with switches, exchanges and transmission equipments. Various devices and software in the network monitor the network continuously, generating a large number of different types of events, which can be related to a particular physical device at a certain level in the network, to some software modules, or to individual phone calls. The fact that an event is recorded does not necessarily mean that there is something wrong. However, technical experts strongly believe that some events might be closely related to each other, though the relationship among events is unknown to them. One type of events, trouble reports (*TR*), is of particular importance. A *TR* is put in by a consultant after a customer calling the company to complain some service difficulties, such as noisy lines, no dialling tones or abnormal busy tones. All *TR* must be investigated by the company, often involving sending engineers to check various parts of the network including the customer's place. The total cost for handling *TR* is very high for the telecommunication company due to a very large number of *TR*. With a database of extensive historical monitoring event data and a comprehensive monitoring network, it is highly desirable to find patterns leading to any types of *TR* within one or a few sensible time intervals. Such patterns, when combined with real-time event data, can help to reduce the cost of processing *TR* (for example, to avoid sending engineers to a customer's place if there is an event pattern suggesting the problem might be with an exchange rather than with customer's equipment or connection). Another major incentive to invest in finding such patterns is that, with such knowledge, the company can improve customer services (for example, to contact customers in an possibly affected area before they lodge complaints).

Our problem of long temporal event sequence data mining can be generalized as follows. Let  $k$  types of time-related events be given. A temporal event sequence is a list of events indexed by timestamp reflecting the time when the event was recorded (note that the intervals between adjacent events are arbitrary). Let  $e$  be a selected type of event called *target event*. Additionally, let time interval  $T$  be given. The problem is to find all frequent patterns of events (from the event sequence) which potentially lead to the occurrences of target event  $e$  within the given time interval  $T$ . We call such kind of pattern *event-oriented pattern* because it has temporal relationship with the target event. Note that the temporal relationship discussed in this paper can be extended to other types (such as after target event) or involve more temporal constraints.

The rest of the paper is organised as follows. Related work is discussed in section 2. We formalize the event-oriented mining problem and give a new pair of definitions for confidence and support in section 3. Algorithm for mining event-oriented patterns is presented in section 4. Experiment results are shown in section 5. We summarize our work in section 6.

## 2 Related Work

Data mining has been a topic of substantial research in the last decade. Various sequence mining problems have been investigated, some in the domain similar to that studied in this paper (for example, sequence mining in telecommunication alarm data sets [6, 2, 3]). A number of studies, such as [5], regard a sequence simply as an order among events, and do not consider the timestamp of each event. The problem of placing time windows to make meaningful sequence segmentation is a key problem in temporal sequence data mining. In [3], while event timestamps are considered, the sliding window is moved at a fixed time interval. This time interval, however, is difficult for the user to specify. We shall allow window placement either starting from or ending with any event in this paper. Another different feature of our work is that we introduce a specific target event type (i.e., target event  $e$ ). Therefore, our goal of mining is to find those patterns related to this special event type rather than finding all frequent patterns. Our unique requirements of differentiating event types and placing windows continuously make it necessary to use different support and confidence definitions than those used in [3]. [7] mines failure plans by dealing with bad plans and good plans separately and define the support in a similar way. But it is applied to a set of sequences rather than a long temporal event sequence. In our approach, after creating the dataset, we can apply [8, 9] to find occurrence patterns and [10–13] to find sequential patterns. [4] discusses some failure prediction issues and applies genetic algorithm to compute patterns. However, our algorithm can provide more accurate and complete results.

## 3 Problem Statement

### 3.1 Event Sequences and Patterns

Let us consider  $k$  event types  $E = \{e_i | i = 1..k\}$ . For a given event type  $e_i \in E$ ,  $D(e_i)$  is the domain of the event type which is a finite set of discrete values. An event of type  $e_i$  can be represented as a triple  $(e_i \in E, a \in D(e_i), t)$  such that  $a$  and  $t$  are *value* and *timestamp* of the event respectively. Assume for any  $e_i$  and  $e_j$  in  $E$ ,  $D(e_i) \cap D(e_j) = \emptyset$  if  $i \neq j$ . It means the value of an event can imply its type. Therefore, we simplify an event as  $(a, t)$  where  $a \in Dom = \bigcup D(e_i)$  and maintain a two level hierarchical structure  $\Gamma$  on  $Dom \cup E$ . A *temporal event sequence*, or *sequence* in short, is defined as  $s = \langle (a_1, t_1), (a_2, t_2), \dots, (a_n, t_n) \rangle$  where  $a_i \in Dom$  for  $1 \leq i \leq n$  and  $t_i < t_{i+1}$  for  $1 \leq i \leq n - 1$ .

As one can see from previous discussions, the definition of “patterns of interest” is very broad [10, 3, 5]. Unlike others, we are only interested in patterns that lead to events with a specific value of an event type. Therefore, a *target event value*  $e$  is given in our pattern discovery problem. Note that, in this paper, we discuss patterns only at the event value level, i.e., every element in a pattern corresponds to an event value. If a pattern includes both event value and event type, the method introduced in [14] can be applied to extend patterns across hierarchical levels.

Considering the pattern itself, in this paper, we only discuss the following two types of patterns:

1. *Existence pattern  $\alpha$  with temporal constraint  $T$*  :  $\alpha$  is a set event values, i.e.,  $\alpha \subseteq Dom$ . Events following this pattern must happen within a time interval  $T$ .
2. *Sequential pattern  $\beta$  with temporal constraint  $T$*  :  $\beta$  is in the format of  $\langle a_1, a_2, \dots, a_l \rangle$  where  $a_i \in Dom$  for  $1 \leq i \leq l$ . Events following this pattern must happen within a time interval  $T$ .

A rule in our problem is in the form of  $r = \{LHS \xrightarrow{T} e\}$ . where  $e$  is a target event value and  $LHS$  is a pattern either of type  $\alpha$  or  $\beta$ .  $T$  is the time interval that specifies not only the temporal relationship between  $LHS$  and  $e$  but also the temporal constraint of pattern  $LHS$ . The rule can be interpreted as: a pattern  $LHS$ , which occurs in interval  $T$ , will lead to occurrence of  $e$  within the same interval.

### 3.2 A Model for Support and Confidence

To evaluate the significance of rules, we use support and confidence. While these two measures have been used in the past by most data mining research, the way in which to give a sensible definition to these two important concepts is non-trivial for the problem investigated in this paper, as we shall explain below.

Traditional concepts for support and confidence are defined for transaction databases [8, 15]. The counterpart of “transaction” in temporal event sequence mining can be defined by using a sliding window [3]. That is, a window of size  $T$  is placed along a sequence, such that each time, the sequence is segmented into a sequence fragment which comprises only the events within the window.

Formally, we define *window* as  $w = [t_s, t_e)$  where  $t_s$  and  $t_e$  are start time and end time of the window respectively. We say the *window size* of  $w$  is  $T$  if  $t_e - t_s = T$ . A *sequence fragment*  $f(s, w)$  is the part of sequence  $s$  which is in window  $w$ , i.e.,  $f(s, w) = \langle (a_{m1}, t_{m1}), (a_{m1+1}, t_{m1+1}), \dots, (a_{m2}, t_{m2}) \rangle$  where  $t_i < t_{i+1}$  for  $m1 \leq i \leq m2 - 1$  and  $f(s, w)$  includes all  $(a_i, t_i)$  in  $s$ , if  $t_s \leq t_i < t_e$ .

While the size of the sliding window is typically given by some domain experts (and often multiple sizes can be given according to users’ interests, generating several sets of data mining results), the way to move the window at a fixed time interval, such as in [3], imposes an artificial subsequence boundary, which may lead to exclusion of some useful patterns that are across such boundaries.

To avoid imposing such artificial sequence segmentation boundaries, we move the window to the next event of interest, i.e., a window always either starts from or ends with an event. Further, windows are placed in different ways for calculating support and confidence.

Let us consider support firstly. In transaction databases, support is used to define the frequency of patterns. In our problem a frequent pattern means the pattern frequently happens in  $T$ -sized intervals before target events. Therefore, the frequent pattern should be generated from the sequence fragments before

target events rather than the whole temporal event sequence. To find frequent patterns, windows are placed using the following strategy: first, we locate all the events whose value is  $e$  in the event sequence  $s$  by creating *timestamp set*  $T^e = \{t \mid (e, t) \in s\}$ . For any  $t_i \in T^e$ , we issue a  $T$ -sized window  $w_i = [t_i - T, t_i]$  and get sequence fragment  $f_i = f(s, w_i)$ , where  $T$  is the predefined window size. The set of these sequence fragments  $D = \{f_1, f_2, \dots, f_m\}$  is called *dataset* of target event value  $e$ .

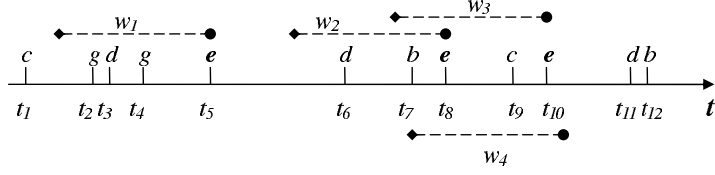


Fig. 1. Example

**Example:** Fig. 1 presents an event sequence. Suppose  $e$  is target event value, the timestamp set of  $e$  is  $\{t_5, t_8, t_{10}\}$ ,  $w_1$ ,  $w_2$  and  $w_3$  are three  $T$ -sized windows ending at timestamp  $t_5$ ,  $t_8$  and  $t_{10}$  respectively. The sequence fragments of these three windows are  $\langle (g, t_2), (d, t_3), (g, t_4) \rangle$ ,  $\langle (d, t_6), (b, t_7) \rangle$  and  $\langle (b, t_7), (e, t_8), (c, t_9) \rangle$ .

The support of rule  $r$  is defined as:

$$supp(r) = \frac{|\{f_i \mid LHS \sqsubseteq f_i\}|}{|D|}$$

In the formula,  $\sqsubseteq$ , called *include*, is a relationship between pattern  $LHS$  and sequence fragment  $f(s, w)$ :

1. For existence pattern  $\alpha$ : Let  $D_f$  be a set of event values that appear in sequence fragment  $f = f(s, w)$ . We denote  $\alpha \sqsubseteq f$  if  $\alpha \subseteq D_f$ .
2. For sequential pattern  $\beta$ : If  $f(s, w) = \langle (a_{m1}, t_{m1}), \dots, (a_{m2}, t_{m2}) \rangle$ , we create a sequence  $s_f = \langle a_{m1}, \dots, a_{m2} \rangle$  by projecting the value of each event in  $f(s, w)$ . We denote  $\beta \sqsubseteq f$  if  $\beta$  is the subsequence of  $s_f$ .

**Example:** In Fig. 1, considering rule  $r = \left\{ \{d\} \xrightarrow{T} e \right\}$ , there are three sequence fragments in dataset  $D$  and two of them ( $f(s, w_1)$  and  $f(s, w_2)$ ) include existence pattern  $\{d\}$ , so  $supp(r) = 66.7\%$ .

With support, the rule can be interpreted as pattern  $LHS$  occurs in a percentage (support) of intervals before target events.

According to the definition of support, given a threshold, we can find all frequent patterns in dataset  $D$ . However, not all the frequent patterns are valuable. For example, a pattern may happen frequently before target events, but this pattern happens everywhere on the time axis, so we still cannot conclude

that this pattern relates to target events. Therefore, confidence is introduced to prune such uninteresting frequent patterns. Intuitively, for each pattern, we need find how many times this pattern happens in sequence  $s$  and out of this number, how many times the occurrence of the pattern leads to the target event.

To count the frequency that a pattern  $LHS$  occurs in sequence  $s$  with respect to window size  $T$ , we introduce the concept *occurrence window*. We think that each occurrence of an interesting pattern corresponds to a  $T$ -sized window. The formal definition of occurrence window is given as follows:

1. For existence pattern  $\alpha$  with temporal constraint  $T$ : We say  $w$  is an occurrence window of the pattern *iff* 1) the size of  $w$  is  $T$ , 2)  $\alpha \sqsubseteq f(s, w)$ , and 3)  $w$  starts from an event of value  $a$  where  $a \in \alpha$ .
2. For sequential pattern  $\beta = \langle a_1, a_2, \dots, a_l \rangle$  with temporal constraint  $T$ : We say  $w$  is an occurrence window of the pattern *iff* 1) the size of  $w$  is  $T$ , 2)  $\beta \sqsubseteq f(s, w)$ , and 3)  $w$  starts from an event of value  $a_1$ .

**Example:** Considering event sequence  $s$  in Fig. 1, window  $w_4$  is an occurrence window of sequential pattern  $\langle b, c, e \rangle$ .

Given sequence  $s$ , pattern  $LHS$ , and time interval  $T$ , we define *occurrence window set* as  $W(LHS, s, T)$ , whose elements are occurrence windows of pattern  $LHS$  with temporal constraint  $T$ .

Confidence of rule  $r$  is defined as:

$$conf(r) = \frac{|W(LHS \rightarrow e, s, T)|}{|W(LHS, s, T)|}$$

The numerator is how many times pattern  $LHS$  plus  $e$  occurs together within  $T$ -sized windows in the sequence  $s$ . The denominator is how many times pattern  $LHS$  itself occurs within  $T$ -sized windows in the sequence  $s$ . Confidence of rule  $r$  indicates that a percentage (confidence) of occurrences of pattern  $LHS$  will lead to events of value  $e$  within a  $T$ -sized interval.

1. For existence pattern  $\alpha$ :  $\alpha \rightarrow e$  means a new pattern in which orders of elements in  $\alpha$  are ignored, but all elements in  $\alpha$  must appear before  $e$ . In this case,  $|W(\alpha \rightarrow e, s, T)|$  is the total number of  $T$ -sized windows that start from any element in  $\alpha$  and include all elements of  $\alpha$  and  $e$  (orders of elements in  $\alpha$  are ignored, but all elements in  $\alpha$  must appear before  $e$ ).
2. For sequential pattern  $\beta = \langle a_1, a_2, \dots, a_l \rangle$ :  $\beta \rightarrow e$  is a new pattern which can be created by simply adding  $e$  as the last element of sequential pattern  $\beta$ , i.e.,  $\beta \rightarrow e \equiv \langle a_1, a_2, \dots, a_l, e \rangle$ .

Now, rule  $r$  with significance measures is in the format of:

$$LHS \xrightarrow{T} e(supp(r), conf(r))$$

### 3.3 Problem Definition

The formal definition of our problem is: Given sequence  $s$ , target event value  $e$ , window size  $T$ , two thresholds  $s_0$  and  $c_0$ , find the complete set of rule  $r = \{LHS \xrightarrow{T} e\}$  such that  $supp(r) \geq s_0$  and  $conf(r) \geq c_0$ .

## 4 Algorithm

In this section we propose an algorithm to solve the problem we defined previously. The basic idea is to generate candidate rules from the part of sequence and prune rules by scanning the whole sequence. High-level structure of the algorithm is shown in Fig. 2.  $R$  is a rule set and initialised as empty. We first create dataset  $D$  which is a set of sequence fragments before target events and then compute a set of frequent patterns whose support is no less than  $s_0$ . After a set of rules is generated according to these frequent patterns, we calculate confidence for each rule in  $R$  and prune those whose confidence is less than  $c_0$ . The last step of the algorithm is to output rules in  $R$ . The main subtasks in this algorithm include 1) creating the dataset, 2) computing frequent patterns and generating candidate rules, and 3) pruning rules. We will explain each subtask in detail. Considering different kinds of pattern formats, the algorithm will be different. In the rest of this section, we use the existence pattern format to demonstrate our algorithm.

<p><b>Input:</b> sequence <math>s</math>, target event value <math>e</math>, window size <math>T</math>, minimum support <math>s_0</math> and minimum confidence <math>c_0</math></p> <p><b>Output:</b> the complete set of rules whose support is no less than <math>s_0</math> and confidence no less than <math>c_0</math></p> <p><b>Method:</b></p> <pre><math>R = \phi;</math> <math>D = d(s, T, e);</math> /* Create the dataset */ <math>R = r(D, s_0, e);</math> /* Find frequent patterns and generate rules */ <math>p(R, s, T, c_0);</math> /* Prune rules */ <b>output</b> <math>R;</math></pre>
--

Fig. 2. High-level structure

### 4.1 Create the Dataset

Fig. 3 shows a query-based approach to creating the dataset. First, a query is issued on sequence  $s$  to locate the  $e$ -valued events and retrieve their timestamps into set  $T^e$ . For each timestamp  $t_i$  in  $T^e$ , we query events whose timestamps are in window  $[t_i - T, t_i)$  to get sequence fragment  $f_i$ . Here  $T$  is the pre-defined window size. Because orders of the events are ignored in existence pattern and also there are no other temporal constraints within a sequence fragment, we use a set  $f'_i$  to only store the values of events in the window. A distinct number is put into initial set to ensure each  $f'_i$  is unique. Finally, we create the dataset  $D$  whose element is  $f'_i$ .

```

Method:  $d(s, T, e)$ 
 $D = \phi; T^e = \phi; id = 0;$ 
compute  $T^e;$ 
for all  $t_i \in T^e$  do
   $id ++;$ 
   $f'_i = \{id\};$ 
  for all event  $(a_k, t_k)$  such that  $t_k \in [t_i - T, t_i)$  do
     $f'_i = f'_i \cup \{a_k\};$ 
   $D = D \cup \{f'_i\};$ 
return  $D;$ 

```

**Fig. 3.** Create the dataset

## 4.2 Compute Frequent Patterns and Generate Candidate Rules

According to the rule format, we know that each rule corresponds to a frequent pattern. So, the key step in this subtask is to find frequent patterns from dataset  $D$ . We can apply algorithm [8,9] for mining existence patterns and [10–13] for sequential patterns.

```

Pattern
   $\{\alpha;$  /* a set of event values */
   $count;$  /* the number of sequence fragments that include  $\alpha$  */ };
Rule
   $\{LHS;$  /* frequent pattern found in  $D$  */
   $RHS;$  /* target event value */
   $|RHS|;$  /* the number of target events */
   $|f_i|LHS \subseteq f_i|;$  /* the number of sequence fragments that include  $LHS$  */
   $supp;$  /* support */
   $|W(LHS, s, T)|;$  /* the number of occurrences of pattern  $LHS$  w.r.t.  $T$  */
   $|W(LHS \rightarrow RHS, s, T)|;$  /* the number of occurrences of  $LHS \rightarrow e$  w.r.t.  $T$  */
   $conf;$  /* confidence */ };

```

**Fig. 4.** Data structures

Data structures of pattern and rule are given in Fig. 4. Fig. 5 shows there are two main steps in this subtask. At the first step, function  $f_r(D, s_0)$  is called to compute frequent patterns and return  $L$ , which is a set of frequent patterns with their support numbers. At the second step, for each frequent pattern, a rule is initialized.

## 4.3 Prune Candidate Rules

The last subtask is to prune the rules whose confidence is less than  $c_0$ . Fig. 6 shows the procedure of this subtask. The basic idea is to scan the sequence

```

Method:  $r(D, s_0, e)$ 
 $L = \phi; R = \phi;$ 
/* Compute frequent patterns */
 $L = f_r(D, s_0);$ 
/* Generate candidate rule set */
for all  $l_i \in L$  do
  Rule  $r_i = \text{new Rule}();$ 
   $r_i.LHS = l_i.\alpha;$ 
   $r_i.RHS = e;$ 
   $r_i.|RHS| = |D|;$ 
   $r_i.|f_i|LHS \sqsubseteq f_i| = l_i.count;$ 
   $r_i.support = \frac{l_i.count}{|D|};$ 
   $r_i.|W(LHS, s, T)| = 0;$ 
   $r_i.|W(LHS \rightarrow RHS, s, T)| = 0;$ 
   $r_i.conf = 0;$ 
   $R = R \cup \{r_i\};$ 
return  $R;$ 

```

**Fig. 5.** Compute frequent patterns and generate candidate rules

```

Method:  $p(R, s, T, c_0)$ 
/* Create a set that include all elements in  $LHS$ s of rules */
 $C = \phi;$ 
for all  $r_i \in R$  do  $C = C \cup r_i.LHS;$ 
/* Count two types of occurrences */
for all  $(a_i, t_i) \in s$  do
  if  $(a_i \in C)$  then
     $w_i = [t_i, t_i + T];$ 
     $f_i = f(s, w_i);$ 
    for all  $r_j \in R$  do
      /* Test patterns */
      if  $(f_i$  starts with the element in  $r_j.LHS$  and  $r_j.LHS \sqsubseteq f_i)$  then
         $r_j.|W(LHS, s, T)| ++;$ 
        if  $(r_j.LHS \rightarrow r_j.RHS \sqsubseteq f_i)$  then
           $r_j.|W(LHS \rightarrow RHS, s, T)| ++;$ 
/* Prune rules */
for all  $r_i \in R$  do
   $r_i.conf = \frac{r_i.|W(LHS \rightarrow RHS, s, T)|}{r_i.|W(LHS, s, T)|};$ 
  if  $(r_i.conf < c_0)$  then  $R = R - \{r_i\};$ 

```

**Fig. 6.** Prune candidate rules

once by using  $T$ -sized slide windows. According to the definition, an occurrence window of any existence pattern must start from the event whose value is an element of the existence pattern. We union all  $LHS$ s of rules to get an event value set  $C$ . Only windows starting from the event, whose value is included in  $C$ , can be occurrence windows of some patterns. When such windows are set,

we test two patterns ( $LHS$  and  $LHS \rightarrow RHS$ ) for each rule. If the sequence fragment in the window includes a pattern, the number of occurrence of this pattern is increased by one. After scanning the sequence once, we can compute the confidence of each rule and prune those whose confidence is less than  $c_0$ .

## 5 Experiments and Discussions

We did experiments on a telecommunication event sequence which consists of 3 event types. Each event type has a number of event values. As we mentioned before, one type of events, trouble reports ( $TR$ ), is of particular importance because it may indicate certain failure on telecommunication networks. Our goal is to find patterns leading to any type of  $TR$ .

**Table 1.** Mining results

minimum support 5% minimum confidence 20%				
Time window size(hours)	6	12	24	48
Number of rules	1	8	52	328

The telecommunication event database contains the data of a state over 2 months. There are 2,003,440 events recorded in the database. Out of this number, 412,571 events are of type  $TR$ . We specify four time intervals and set the thresholds of support and confident as 5% and 20% respectively. The mining results are given in Table 1. An example of rule is shown in Table 2. This rule means signal  $f\_1$  and  $e\_1$  together will lead to a trouble report  $NTN$  within 24 hours with some probabilities.

**Table 2.** Rule example

T (hours)	LHS	RHS	Support	Confidence
<b>24</b>	<b>f_1 e_1</b>	<b>NTN</b>	<b>6.32%</b>	<b>38.72%</b>

In the experiment, we need to find interesting patterns for each value of  $TR$ . It means a target event value is extended to a set of target event values. In this case, for each target event value, candidate rules are generated. We then combine these rules and scan the sequence once to determine their confidence.

In the second subtask, note that minimum support  $s_0$  is a percentage. For a given target event value, the minimum support number should be  $m * s_0$ , where  $m$  is the population of the events with this target event value. To ensure the patterns have statistic significance, we define a threshold  $m_0$  which is minimum population of target event value. We create the dataset for a target event value

only if its population exceeds this threshold. In this case, the minimum support number of any target event value is set as  $m_0 * s_0$ .

Considering a set of target event values, some rules may have the same *LHS*. We can group rules according to their *LHS* and create a list for each group. The head of the list is the *LHS* of this group and linked by the *RHS* of each rule in this group. For example, if a group includes rules  $\alpha \rightarrow e_1$ ,  $\alpha \rightarrow e_2$ , and  $\alpha \rightarrow e_3$ , we create a list which is shown in Fig. 7.



**Fig. 7.** Example of the list

Given a window, for each list, we first test whether a window includes the head of this list (which is the pattern *LHS*), if not, the test of this list terminates; otherwise, we record the last element which meets the requirement of the *LHS* and test whether the rest elements in the window include each *RHS*. We can also remove some lists if the heads of these lists are super patterns of a *LHS* which has been proved as not included in the window.

In the implementation, we store dataset  $D$  in the main memory. This becomes hard if the sequence is very long. To solve this problem, the sequence is divided into some segments such that the dataset of each segment can be fitted in the memory. We can first find frequent patterns from these segments separately, then combine these local frequent patterns together and scan the entire sequence once to determine their support. The candidate rules are generated only for the patterns whose global support is no less than the threshold. The principle of this approach is: if a pattern is frequent to the whole sequence, it must be frequent to at least one segment. [16] discusses this idea in detail. Because the fragmentation will break down the patterns near the borders of segments, this approach can improve efficiency but sacrifice some accuracy.

## 6 Summary

This paper generalizes the problem of discovering event-oriented patterns from a long temporal event sequence. We introduce speciality of events and give our definitions of support and confidence. An algorithm based on a window placement strategy is proposed to solve this problem.

Future research includes data management issues and performance issues. In this paper, we only discuss existence pattern and sequential pattern. Other pattern formats can also be computed based on our basic idea. More temporal constraints can be added into mining process. The problems of how to select the appropriate window size and evaluate rules should also be investigated. For performance issues, as a query-based approach to creating the dataset is not

efficient enough, the improvement of the efficiency of the algorithm is currently under the consideration.

## Acknowledgement

X. Sun would like to thank Dr. Xue Li for helpful discussions.

## References

1. Wang, J.T.L., Chirn, G.W., Marr, T.G., Shapiro, B.A., Shasha, D., Zhang, K.: Combinatorial pattern discovery for scientific data: Some preliminary results. In: Proc. 1994 ACM SIGMOD Intl. Conf. on Management of Data. (1994) 115–125
2. Mannila, H., Toivonen, H.: Discovering generalized episodes using minimal occurrences. In: Knowledge Discovery and Data Mining. (1996) 146–151
3. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery* **1** (1997) 259–289
4. Weiss, G.M., Hirsh, H.: Learning to predict rare events in event sequences. In: Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD'98), New York, NY, AAAI Press, Menlo Park, CA (1998) 359–363
5. Yang, J., Wang, W., Yu, P.S.: Infominer: mining surprising periodic patterns. In: Proc. 7th ACM SIGKDD Conference. (2001) 395–400
6. Hatonen, K., Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H.: Knowledge discovery from telecommunication network alarm databases. In: Proc. 12th International Conference on Data Engineering. (1996) 115–122
7. Zaki, M.J., Lesh, N., Ogihara, M.: Planmine: Sequence mining for plan failures. In: Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD'98), New York, NY, ACM Press (1998) 369–373
8. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. 20th Int. Conf. Very Large Data Bases, Morgan Kaufmann (1994) 487–499
9. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proc. 2000 ACM SIGMOD Intl. Conf. on Management of Data, ACM Press (2000) 1–12
10. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: Proc. 5th Int. Conf. Extending Database Technology. Volume 1057., Springer-Verlag (1996) 3–17
11. Zaki, M.J.: SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning* **42** (2001) 31–60
12. Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.: Freespan: Frequent pattern-projected sequential pattern mining. In: Proc. 6th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, ACM Press (2000) 355–359
13. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: PrefixSpan mining sequential patterns efficiently by prefix projected pattern growth. In: Proc. 2001 Int. Conf. Data Engineering, Heidelberg, Germany (2001) 215–226
14. Srikant, R., Agrawal, R.: Mining generalized association rules. *Future Generation Computer Systems* **13** (1997) 161–180
15. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proc. 11th Int. Conf. on Data Engineering, Taipei, Taiwan, IEEE Computer Society Press (1995) 3–14
16. Savasere, A., Omiecinski, E., Navathe, S.B.: An efficient algorithm for mining association rules in large databases. In: *The VLDB Journal*. (1995) 432–444