Sequence Prediction using Partially-Ordered Episode Rules

Yangming Chen
School of Computer Sciences
Harbin Inst. of Technology (Shenzhen)
Shenzhen, China
510717841@qq.com

Philippe Fournier-Viger*
School of Humanities and Social Sciences
Harbin Inst. of Technology (Shenzhen)
Shenzhen, China
philfv8@yahoo.com

Farid Nouioua
University of Bordj Bou Arreridj
Bordj Bou Arreridj, Algeria
faridnouioua@gmail.com

Youxi Wu
Hebei University of Technology
Tianjin, China
wuc567@163.com

***Abstract*—Predicting the next event (or symbol) of a sequence has many applications such as network event prediction, webpage prefetching, activity recommendation and stock price prediction. An effective approach for event prediction that has the benefit of being interpretable, is to extract episode rules to then apply them for prediction. This paper improves upon this approach by using a novel type of episode rules called partially-ordered episode rules. These rules are more general than standard episode rules as they loosen the ordering constraint on events in each rule antecedent and consequent. Hence, a partially-ordered episode rule can replace multiple standard episode rules. Experiments on several benchmark datasets show that partially-ordered episode rules can provide more accurate predictions than standard episode rules.**

***Index Terms*—Sequence Prediction, Episode rules, Partially Ordered Episode Rules.**

I. Introduction

Predicting the future is key to decision-making. To perform accurate predictions, it is necessary to consider previous observations and their temporal relationships with possible upcoming observations. Temporal observations are typically represented as either time series or discrete sequences [1]. A *time series* is an ordered list of numbers, generally measured at a fixed time interval. Predicting the next observation of a time series is typically viewed as a problem of finding a function that closely fit the data points. This can be done with techniques such as least square linear regression or more complex techniques. On the other hand, a *discrete sequence* is an ordered list of symbols. In this paper, the focus is on discrete sequences as their prediction is useful in many domains. For instance, it can be used to predict the next word that a user will type on a phone, the next error that will occur in a network, the next purchase of a customer, and the next location where someone will drive [2]. Because the nature of discrete sequences is much different than time series, different techniques are used for predicting the next symbol of a discrete sequence than for time series. One of the most accurate techniques for event prediction is artificial neural networks. However, a major drawback is that they mostly operate as black-boxes. Thus, a user is often unable to understand the reasons why an event is predicted. But developing explainable models is often critical for decision-makers in the industry. This is for example the case for network fault management, where network technicians wish to not only predict network errors but understand the relationships between complex network events to prevent errors [3].

A promising technique that can provide good prediction accuracy while being also easily interpretable by humans is to extract episode rules from a discrete sequence [4]–[6], to then use the rules for making predictions. A traditional episode rule has the form $X \rightarrow Y$ indicating that if a sequence of event(s) X appears, it will be likely followed by another sequence of event(s) Y. For example, a rule $\langle a, b, c\rangle \rightarrow \langle d\rangle$ means that if event a is followed by b and then c, it will be followed by d. Though, such rules are easily understood by users, a drawback is that these rules are too specific. For example, the above rule is treated as different from many similar rules such as $\langle b, c, a\rangle \rightarrow \langle d\rangle$, $\langle c, a, b\rangle \rightarrow \langle d\rangle$, $\langle c, b, a\rangle \rightarrow \langle d\rangle$ $\langle a, c, b\rangle \rightarrow \langle d\rangle$ and $\langle b, a, c\rangle \rightarrow \langle d\rangle$. But in practice, all these rules are similar and may represent the same situation where d is following the events a, b and c (in any order). Because many rules may have only small ordering variations, many redundant rules may be presented to the user.

To address this problem, this paper proposes to replace traditional episode rules for event prediction by a novel type of rules called partially-ordered episode rules (POER) [7]. These rules have the form $X \rightarrow Y$, indicating that if some set of event(s) X appear in any order, they will be followed by another set of event(s) Y in any order. Because this definition loosen the ordering constraint of traditional episode rules, a single POER can replace multiple traditional episode rules. For instance, the POER $\{a, b, c\} \rightarrow \{d\}$ can replace the six rules presented above. This paper presents a comparison of the POER with two types of traditional episode rules on several

*Corresponding author.

benchmark datasets. It is found that POER can provide more accurate predictions than the compared traditional episode rules.

The rest of this paper is divided into four sections. Section 2 provides a formal definition of the problem of event sequence prediction. Section 3 describes the three types of episode rules compared in this study. Section 4 introduces the designed testing framework, called EpisodePredictor, for comparing the three types of rules. Section 5 presents the results of the experiments. Finally, Section 6 draws a conclusion and discusses future work.

II. The Problem of Event Sequence Prediction

The problem of predicting the next event of a sequence is defined based on the following definitions. Let there be a finite set of *events* $E = \{e_1, e_2, \ldots e_n\}$, indicating all possible event types (or symbols) that may appear over time. An **event set** R is a subset of events, i.e. $R \subseteq E$. An **event pair** $P = (t, R)$ is composed of a timestamp t (positive integer) and a non empty event set R. A **sequence** $S = \langle (t_1, R_1), (t_2, R_2), \ldots, (t_m, R_m) \rangle$ is an ordered list of event pairs, where $0 \leq t_1 < t_2 < \ldots < t_m$ and $R_i \subseteq E$ for $1 \leq i \leq m$.

Example 1: Consider the set of events $E = \{a, b, c, d, e\}$. The sequence $\langle (1, \{a\}), (2, \{b, c\}), (4, \{d\}), (7, \{e\}) \rangle$ indicates that an event a occurred at time 1, followed by events b and c at time 2, then followed by d at time 4, and finally by event e at time 7.

The *problem of predicting the next event of a sequence* is defined as follows.

Definition 1 (Sequence prediction): Let there be two user-defined positive numbers α and γ called the **prefix length** and **suffix length**, respectively. Moreover, let there be a sequence $S = \langle (t_1, R_1), (t_2, R_2), \ldots, (t_v, R_v), \ldots (t_w, R_w) \rangle$, where the events $R_1, R_2, \ldots, R_v$ have been observed and all events after t_v have not occurred yet. The **prefix** of S is the subsequence consisting of all event pairs of S occurring in the time interval $[t_{v-\alpha}, t_v]$. The **suffix** of S is the subsequence consisting of all event pairs of S occurring in the time interval $(t_v, t_{v+\gamma}]$. **Sequence prediction** consists of predicting an event that will appear in the time interval $(t_v, t_{v+\gamma}]$, that is in the *suffix* of S, based on events from the *prefix*.

Example 2: Consider the event sequence $\langle (1, \{a\}), (2, \{b\}), (3, \{c\}), (4, \{d\}), (5, \{e\}), (6, \{a\}) \rangle$ occurring on a computer network. For $\alpha = 1$, $\gamma = 2$ and $v = 3$, the task of sequence prediction consists of predicting an event from $\langle (4, \{d\}), (5, \{e\}) \rangle$ given the events $\langle (2, \{b\}), (3, \{c\}) \rangle$.

III. Three Types of Episode Rules

To be able to predict the next event of a sequence using episode rules, the first step consists of extracting episode rules from a training sequence. This can be done by applying an episode rule mining algorithm. The result is a set of episode rules. Then, for a new sequence S, the prefix of that sequence is compared with the rules to predict the next event. The following paragraphs describes three types of rules: the standard episode rules, the precise-positioning episode rules and the proposed partially-ordered episode rules.

A. Standard Episode Rules

The traditional type of episode rules, hereafter called *standard episode rules* (SER) are obtained in two steps. First, an episode mining algorithm [5], [6], [8] is applied to identify frequent sequence of events, called composite episodes. Then, pairs of those episodes are combined to build episode rules [5], [6] indicating strong relationships between episodes. A formal definition is given next.

Definition 2 (Composite episode): A **composite episode** ξ is a sequence of event sets having the form $\langle Y_1, Y_2, \ldots, Y_p \rangle$ where each event set $Y_i \subseteq E$ is said to precede another event set Y_j if $i < j$.

Definition 3 (Serial episode): A **serial episode** is a composite episode where each event set contains a single event.

Episodes are designed to capture sequential ordering relationships between events in a sequence. It can be observed that they do not contain timestamps so as to be more general. As a result, an episode can appear multiple times in a sequence at different timestamps. Episode mining algorithms identify frequent episodes by counting their support (number of occurrences) in the input sequence. There are different ways to calculate the support of an episode in a sequence. In previous studies, it was argued that the *head frequency* support is the most useful one [5], [8], [9]. The head support measure is defined based on a concept of occurrence.

Definition 4 (Occurrence): Let there be an episode $\xi = \langle Y_1, Y_2, \ldots, Y_p \rangle$ and a sequence $S = \langle (t_1, R_1), (t_2, R_2), \ldots, (t_v, R_v) \rangle$. A time interval $[t_s, t_e]$ is an **occurrence** of ξ in S if there are integers $t_s = z_1 < z_2 < \ldots < z_w = t_e$ such that $Y_1 \subseteq R_{z_1}$, $Y_2 \subseteq R_{z_2}$, $\ldots$, $Y_p \subseteq R_{z_w}$. The timestamps t_s and t_e are said to be the **start point** and **end point** of an occurrence $[t_s, t_e]$. The notation $occ(\xi, S)$ refers to the set of **all occurrences** of ξ in S.

Definition 5 (Head support): The **head support** of an episode ξ in a sequence S is defined as the number of distinct start points that are in $occ(\xi, S)$. Formally, $sup(\xi, S) = |\{t_s | [t_s, t_e] \in occ(\xi)\}|$ *[5].*

A traditional episode mining algorithm such as MINEPI+ [5] extracts all frequent composite episodes, that is episodes having a support greater or equal to a user-defined threshold $minsup > 0$ in a sequence S. To filter out very long occurrences, a maximum occurrence length constraint $winlen > 0$ can be specified by the user. In that case, all occurrences longer than $winlen$ are ignored.

Example 3: Fig. 1 depicts a sequence S having 11 timestamps. The composite episode $\xi = \langle \{a\}, \{d\} \rangle$ has three occurrences in S, that are $occ(\xi, S) = \{[t_2, t_3], [t_2, t_8], [t_5, t_8]\}$. That episode has a support $sup(\xi, S) = 2$ since its occurrences have two distinct start points (t_2 and t_5). If $winlen = 2$, then $sup(\xi, S) = 1$ since occurrence $[t_5, t_8]$ is discarded.

After discovering frequent composite episodes in a sequence, a post-processing algorithm can be applied to generate episode rules [5], [6]. A **standard episode rule** is

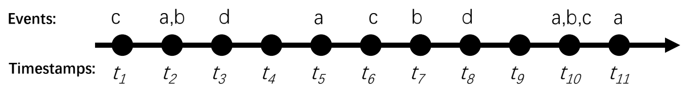

Fig. 1: An event sequence with 11 timestamps

an implication of the form $\xi \rightarrow \zeta$ between two frequent composite episodes ξ and ζ. To find interesting episode rules, the confidence of each potential episode rule is calculated as $conf(\xi \rightarrow \zeta, S) = sup(\xi \rightarrow \zeta, S)/sup(\xi, S)$ where $sup(\xi \rightarrow \zeta, S)$ represents the number of occurrences of ζ appearing after ξ in time intervals no greater than $winlen$. Only the episode rules that have a confidence that is no less than a user-defined $minconf$ threshold are kept. These rules can then be used for sequence prediction.

Example 4: Consider the sequence of Fig. 1, $winlen = 2$, $\xi = \langle\{a\}\rangle$ and $\zeta = \langle\{d\}\rangle$. It can be found that $sup(\xi, S) = 4$ and $sup(\xi \rightarrow \zeta, S) = 1$. Hence, $conf(\xi \rightarrow \zeta, S) = \frac{1}{4}$.

B. Precise-Positioning Episode Rules

Recently, another type of traditional episode rules called *precise-positioning episode rules* (PER) was proposed [4], which is also used as baseline in this study. The key difference between SER and PER is that the latter contains additional information about the time elapsed between events so as to provide more accurate predictions. The concept of PER is defined as follows.

Definition 6 (Fixed-Gap Episode): A **fixed-gap episode** ζ is an ordered list of pairs having the form $\zeta = \langle(\Delta_0, Y_1), (\Delta_1, Y_2), (\Delta_1, Y_3), \ldots, (\Delta_{k-1}, Y_k)\rangle$. Each pair (Δ_{i-1}, Y_i) for $i \in [1, k]$ is composed of an event set $Y_i \in E$ and a time duration Δ_{i-1} indicating the elapsed time between the observation of Y_i and that of the previous event set Y_{i-1}. Note that by definition, $\Delta_0 = 0$.

Example 5: The fixed-gap episode $\langle(0, \{c\}), (2, \{d\})\rangle$ indicates that event d appeared two time units after c. It appears in the sequence of Fig. 1 in $[t_1, t_3]$.

Definition 7 (Precise-Positioning Episode Rule): A **precise-positioning episode rule** (PER) is an implication $\xi \xrightarrow{\Delta_t} \zeta$ where $\xi = \langle X_1, X_2, \ldots, X_p\rangle$ is a frequent serial episode while $\zeta = \langle(\Delta_0, Y_1), \ldots, (\Delta_{k-1}, Y_k)\rangle$ is a fixed-gap episode. Each PER must satisfy three conditions. First, ξ must be a frequent episode given a user-defined maximum window size $winlen$. Second, the elapsed time between X_p and Y_1 must be equal to a user-defined constant Δ_t. Third, the value $q = \sum_{i=1}^{k-1} \Delta_i + \Delta_t$ must satisfy $q \leq \gamma$ for a user-defined time span γ.

Example 6: For sequence S of Fig. 1 and $\gamma = 5$, the PER $\langle\{c\}\rangle \xrightarrow{2} \langle(0\{a, b\}), (1, \{d\})\rangle$ has an occurrence in $[t_1, t_3]$ of S. This rule means that if c appears, two time units later, a and b will occur, followed by d, one time unit later.

The PER can be extracted from an event sequence using an efficient algorithm such as PRU [4]. To filter out irrelevant PER, PRU calculates the support and confidence of each PER and discards those having a support or confidence that is no less than user-defined $minsup$ and $minconf$ thresholds, respectively. The remaining rules can then be used for sequence prediction [4].

C. Partially-Ordered Episode Rules

The third type of episode rules compared in this paper are partially-ordered episode rules [7], which the authors have recently proposed. POER are said to be *partially-ordered* because the constraint of SER that events inside a rule antecedent (or consequent) must be ordered is removed, while the constraint that the antecedent must precede the consequent is preserved. This allows finding episode rules that are more general than SER and PER.

Definition 8 (Partially-ordered episode rule): Let there be two non empty event sets $Y_1, Y_2 \subseteq E$. A **partially-ordered episode rule** (POER) is an implication of the form $Y_1 \rightarrow Y_2$. A POER $Y_1 \rightarrow Y_2$ is interpreted as if all events in Y_1 are observed in any order, they will be followed those in Y_2 in any order.

To find interesting POER, the support and confidence measures are used as well. These measures are defined as follows for POER.

Definition 9 (POER occurrence): Let there be three user-defined temporal constraints, namely $\alpha, \eta, winlen \in \mathbb{Z}^+$. An **event set** $Y \subseteq E$ **occurs in a time interval** $[t_i, t_j]$ of a sequence $S = \langle(t_1, R_1), (t_2, R_2), \ldots, (t_v, R_v)\rangle$ if $Y \subseteq R_i \cup R_{i+1} \ldots \cup R_j$. Furthermore, a **POER** $Y_1 \rightarrow Y_2$ **occurs in a time interval** $[t_i, t_j]$ of S if there are timestamps t_v, t_w such that $t_i \leq t_v < t_w \leq t_j$, Y_1 has an occurrence in $[t_i, t_v]$, Y_2 has an occurrence in $[t_w, t_j]$, $t_v - t_i < \alpha$, $t_j - t_i < winlen$, and $t_j - t_w < \eta$.

Definition 10 (POER support): The **head support of a POER** $Y_1 \rightarrow Y_2$ in a sequence S is defined as $sup(Y_1 \rightarrow Y_2, S) = |occ(Y_1 \rightarrow Y_2, S)|$ where $occ(Y_1 \rightarrow Y_2, S)$ is the number of occurrences of the rule that have distinct start points.

Definition 11 (POER confidence): The **confidence** of a rule $Y_1 \rightarrow Y_2$ is defined as $conf(Y_1 \rightarrow Y_2, S) = |occ(Y_1 \rightarrow Y_2, S)|/|occ(Y_1, S)|$, where $occ(Y_1, S)$ is the number of occurrences of the event set Y_1 that have distinct start points.

We designed the POERM algorithm [7] to efficiently discover all the POER in a sequence S. It requires setting the α, η and $winlen$ constraints, as well as two thresholds $minsup$ and $minconf$. The output is each POER r satisfying the conditions $sup(r, S) \geq minsup$ and $conf(r, S) \geq minconf$.

Example 7: Let there be $minsup = 3$, $minconf = 0.6$, $\alpha = 3$, $winlen = 4$, $\eta = 1$ and the sequence S of Fig. 1. The event set $\{a, b, c\}$ has three occurrences: $occ(\{a, b, c\}, S) = \{[t_1, t_2], [t_5, t_7], [t_{10}, t_{10}]\}$ The rule $r : \{a, b, c\} \rightarrow \{d\}$ has two occurrences: $occ(r, S) = \{[t_1, t_3], [t_5, t_8]\}$. Hence, $supp(r, S) = 3$, $conf(r, S) = 2/3$, and R is output as a POER.

In this section, we first introduce key definitions for cross-level HUIM. Then, we give a mathematical formulation of the novel problem of top-k cross-level HUIM.

IV. The Testing Framework

This section describes the designed framework, called EpisodePredictor, for comparing the prediction accuracy of different types of episode rules. The framework is used in this paper to compare the two traditional types of episode rules (SER and PER) with the proposed POER. The framework is applied in two phases.

1) Training. In that phase, episode rules are extracted from the training data. The user must provide an event sequence S to be used for training as well as values for the $minsup$, $minconf$, $winlen$, α and γ parameters. Only episode rules containing a single item in the consequent are extracted as those are sufficient for next event prediction. If the user chooses to extract SER, the Minepi+ [5] algorithm is applied. If the user chooses to extract PER, the PRU [4] algorithm is utilized where Δ_t is set as $\Delta_t = \gamma$. And if the user decides to extract POER, the POERM [7] algorithm is utilized where η is set as $\eta = \gamma$.

2) Prediction. In the second phase, the user must provide a test sequence S' to evaluate the predictions obtained using the rules. First, a prefix window of length α is slide over the test sequence. For each position of that window, an event following the window is predicted using the rules. This is accomplished by calling Algorithm 1. That prediction algorithm takes as input a prefix window P and a set of episode rules. The algorithm performs two steps. Step 1 is to scan all the episode rules to find those matching the prefix window. A rule is said to match with a prefix if the antecedent appears in the prefix. Step 2 is to calculate a prediction based on the matching rules. This is done by choosing one of the matching episode rules. Various criteria could be used for this purpose. After some preliminary tests, it was found that selecting the rule having the highest score provided the best predictions, where the score of a rule r is $Score(r) = (c_1 \times conf(r, S) + c_2 \times sup(r, S)) \times l(r, P)$, where c_1 and c_2 are preset values and $l(r, P)$ is the number of events in r that match with P. The constants c_1 and c_2 were empirically set as $c_1 = 0.7$ and $c_2 = 0.3$ in the experiments presented in this paper but other values may be more suitable for other data. Following the selection of the rule having highest score, the prediction is made by choosing the event in the rule consequent. In the case, where no rule is found to match with the prefix P, the prediction is $null$, meaning that no prediction can be made.

There are thus three possible outcomes for a prediction: (1) if the predicted event is found in the suffix following the prefix P, it is a *good prediction*, (2) if the predicted event is not found, it is a *bad prediction*, and (3) if no prediction is made, then it is a *no match*.

Algorithm 1: The Prediction algorithm

```
input : $P$: a prefix window, $Rules$: a set of episode
        rules
output: the predicted event
1  $c1 \leftarrow 0.7$;
2  $c2 \leftarrow 0.3$;
3  $prediction \leftarrow null$;
4  $bestScore = 0$;
5  foreach each episode rule $r \in Rules$ do
6      Calculate $l(r, P)$;
7      Retrieve $conf(r, S)$ and $sup(r, S)$;// were
           calculated during training
8      $score \leftarrow (c_1 \times conf(r, S) + c_2 \times sup(r, S)) \times l(r, P)$;
9      if $score > bestScore$ then
10         $bestScore \leftarrow score$;
11         $prediction \leftarrow$ consequent of $r$;
12     end
13 end
14 return $prediction$;
```

V. Experimental Evaluation

The experimental evaluation was done using four public datasets often used in episode and pattern mining studies, which can be downloaded from the SPMF pattern mining library [10]:

- The *Bible* dataset is a transformation of the *Bible* into an event sequence where each word is an event. The sequence has 649,024 events (words) selected from 13,905 distinct words (event types).
- The *OnlineRetail* dataset is a sequence of 541,909 transactions from a UK-based online store, containing 2,364,798 events representing the purchase of items from 2,603 distinct event (item) types.
- The *FIFA* dataset is a sequence of 710,435 click events on the FIFA World Cup 98 website, where there are 2,990 distinct event types.
- The *Leviathan* dataset is a sequence of words obtained by transforming the Leviathan novel written by Thomas Hobbes (1651). The sequence has 153,682 words from 9,025 distinct word types.

Each dataset was divided into two parts: a **training set** (the first 75% of a sequence) and a **testing set** (the remaining 25% of the sequence). The training set is used to generate episode rules, while the testing set is utilized to evaluate predictions using these rules. During the evaluation, statistics were recorded about the number of correct predictions made by each type of rules and the total number of predictions opportunities. Using this information two measures were calculated. The **accuracy** is the number of good predictions divided by the number of prediction opportunities (the number of windows in the test set). The **matching rate** is the number of good or bad predictions divided by the number of prediction opportunities. For instance, consider that there are 1000 prediction opportunities (windows), that a model makes 500 good predictions, 200 bad predictions and cannot make predictions for 300 windows since no rules are matching. In that case, the accuracy is 500 / 1000 = 0.5 (50%) and the matching rate is 700 / 1000 = 0.7 (70%).

Preliminary tests were done to fine tune each model's parameters to obtain the best accuracy and matching rate. Based on these tests, the default values for ($minsup$, $minconf$, α,

γ) on the *Bible*, *OnlineRetail*, *FIFA* and *Leviathan* datasets are (120, 0.3, 4, 1), (650, 0.3, 4, 1), (2800, 0.3, 2, 2) and (80, 0.1, 2, 2), respectively. From these default values, several experiments are presented next where a single parameter is varied in each experiment to assess its influence.

A. Influence of $minsup$

The first experiment consisted of varying $minsup$ to assess its influence on the accuracy and matching rate for the three types of episode rules. The POERM, Minepi+ and PRU algorithms were used to extract the POER, SER and PER, respectively. The range of $minsup$ values and results for the four datasets are shown in Fig. 2.

The can be observed that compared to precise-positioning episode rules, partially-ordered episode rules can improve accuracy by up to 4% and matching rate by up to 34%. And compared to SER, POER can improve accuracy by up to 11% and the matching rate by up to 18%. This shows that using partially-ordered episode rules is beneficial on the four datasets.

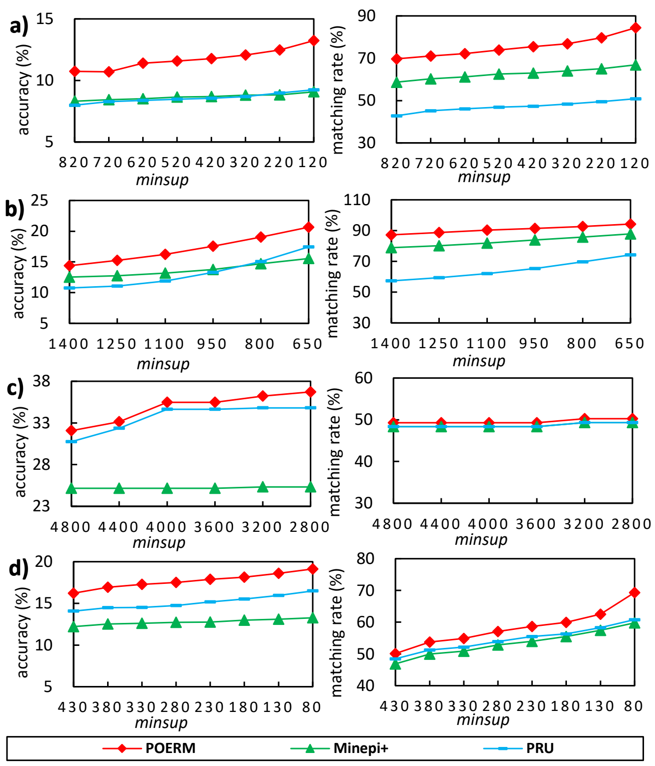

Fig. 2: Influence of $minsup$ on (a) Bible, (b) OnlineRetail, (c) FIFA and (d) Leviathan

B. Influence of $minconf$

The second experiment consisted of varying $minconf$ to assess its influence on the accuracy and matching rate. Fig. 3 shows the results obtained by varying $minconf$ from 0.3 to 0.8 for the four datasets. These results show that partially-ordered episode rules can improve accuracy by up to 4% and matching rate by up to 34% compared to precise-positioning episode rules and improve accuracy by up to 11% and matching rate by up to 18% compared to standard episode rules.

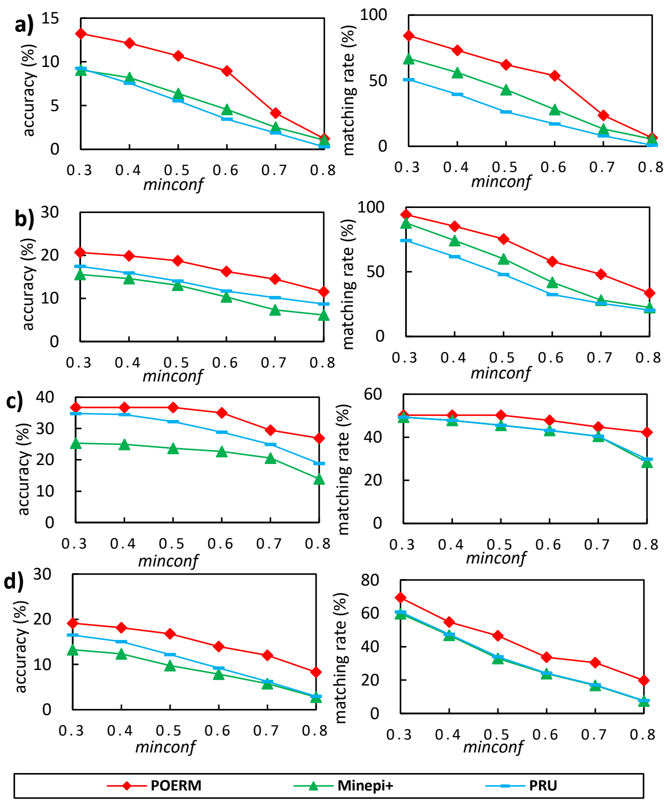

Fig. 3: Influence of $minconf$ on (a) Bible, (b) OnlineRetail, (c) FIFA and (d) Leviathan

C. Influence of the prefix length α

The third experiment consisted of varying prefix length α (the number of preceding itemsets that are used for making a prediction) to assess its influence on the accuracy, matching rate and rule count. Fig. 4 respectively show the results obtained by varying this parameter from 2 to 7 on the four datasets. It is observed that α has a strong influence on results. Increasing α generally results in more rules, which increases the matching rate. But at some point increasing the prefix size has less and less influence, which is reasonable. For the accuracy, it can decrease when the prefix length is increased but this is due to the increase in matching rate. It is seen that partially-ordered episode rules improved accuracy by up to 4% and matching rate by up to 34% compared to precise-positioning episode rules and by up to 11% and 18% compared to standard episode rules.

D. Influence of the suffix length γ

The fourth experiment consisted of varying the suffix length γ (number of itemsets that should be considered for making a prediction) to assess its influence on the accuracy. Fig. 6 shows the results obtained by varying this parameter from 1 to 7 for

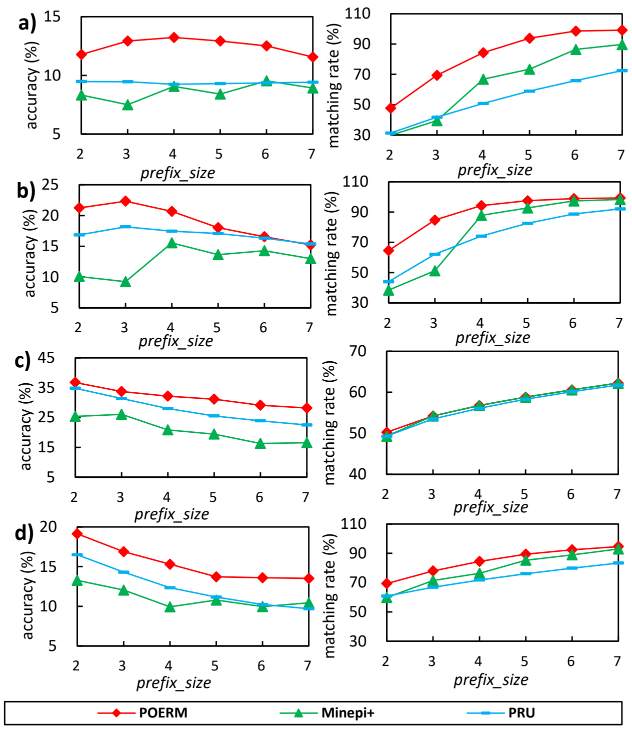

Fig. 4: Influence of $prefix\_size$ on (a) Bible, (b) OnlineRetail, (c) FIFA and (d) Leviathan

the four datasets. In general, increasing γ also increases the accuracy and matching rate, which is reasonable as it gives more chance to make a good prediction. It is observed that partially-ordered episode rules improved accuracy by up to 40% and matching rate by up to 40% compared to precise-positioning episode rules and improved accuracy by up to 13% and matching rate by up to 2% compared to standard episode rules.

E. Comparison of the number of rules obtained by the three algorithms

The number of rules generated by the three algorithms was also measured. Fig. 5 shows the number of episode rules of the three types when $minsup$ is varied. It can be observed that the number of rules for the three types of rules varies differently because their definitions are different. For three out of four datasets, the number of POER is higher than that of SER and PER. This may seem counter-intuitive because POER are more general than SER and PER, and thus a POER can replace several SER and PER. However, since POER are more general than SER and PER, they can match more often with a sequence, and thus have a higher support and confidence. For this reason, some POER may be discovered while the corresponding SER and PER may not be found. This phenomenon can be observed for the toy sequence of Fig. 1, $minsup = 2$ and $winlen = 4$. While the SER $\langle\{a\},\{c\}\rangle \rightarrow \langle\{d\}\rangle$ and $\langle\{c\},\{a\}\rangle \rightarrow \langle\{d\}\rangle$ each have a support of 1, the POER $\{a,c\} \rightarrow \{d\}$ has a support of 2.

TABLE I: Comparision of the three algorithm on Runtime and Max Memory Usage

Dataset	Runtime(s)			Max Memory Usage(mb)		
	POERM	Minepi+	PRU	POERM	Minepi+	PRU
Bible	3.03	48.26	4.34	416.3	114.6	962.6
OnlineRetail	12.03	141.84	16.76	1,196.9	73.8	1,606.7
FIFA	2.20	5.39	2.20	636.3	87.1	373.9
Leviathan	4.75	4.24	3.38	959.8	51.2	773.5

Hence, only the latter rule is found. This explains why more POER can be found in some cases compared to SER and PER.

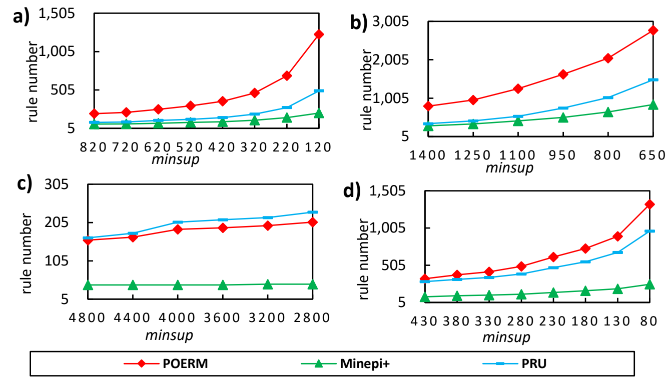

Fig. 5: Influence of $minsup$ on rule numbers on (a) Bible, (b) OnlineRetail, (c) FIFA and (d) Leviathan

F. Runtime and memory efficiency

Tab. I shows the maximum runtime and memory consumption of the three algorithms POERM, Minepi+ and PRU for the four datasets. For training (extracting episode rules), it was found that in general, POERM runs slightly faster than PRU, and both are much faster than Minepi+. In terms of maximum memory usage, POERM and PRU consume more memory than Minepi+. However, the amount of memory is still small, given that current workstations are often equipped with tens of gigabytes of RAM. Based on these results, extracting rules with POERM can be viewed as efficient. And although, it is not always the most efficient, it can provide an increased accuracy and matching rate, which can be viewed as a good trade-off. It should also be noted that rule extraction can be performed once and then multiple predictions can be made using the rules.

In terms of runtime for making predictions, it was found that there is no major differences between the three types of rules. This is because although the three algorithms produces different types of rules, they utilize the same process for matching rules to a sequence to make a prediction (Algorithm 1). Using the three types of rules, over 3,000 predictions per second could be done on the test datasets, which can be considered as a very satisfying performance.

G. Summary

On overall, results show that using the proposed POER in place of SER and PER can provide great benefits for sequence prediction in terms of accuracy and matching rate, and that POERM is generally very efficient compared to PRU and Minepi+.

It should be noted that a higher accuracy might be obtained using machine learning models such as artificial neural networks. However, those models are generally black-boxes. A benefit of discovering episode rules is that they are easily interpretable, which is often required by decision-makers for real applications in the industry such as for network fault management [3].

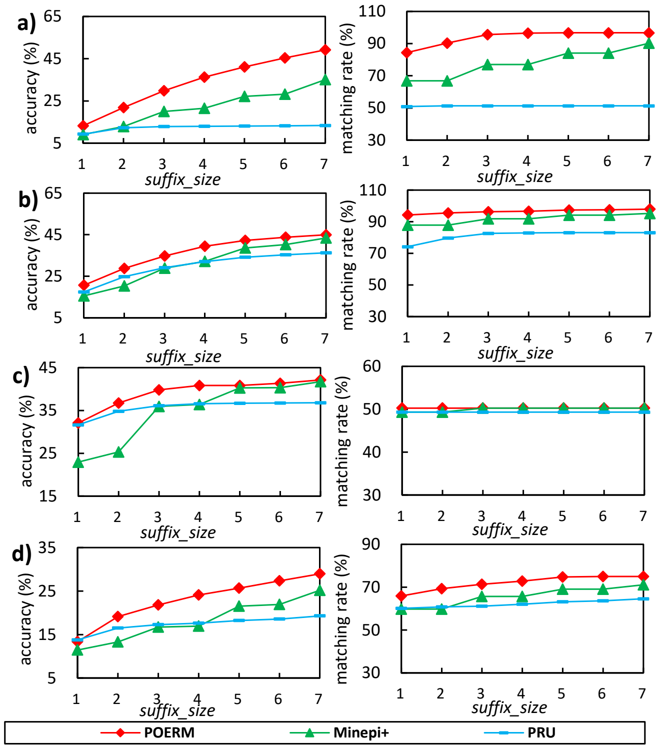

Fig. 6: Influence of $suffix\_size$ on (a) Bible, (b) OnlineRetail, (c) FIFA and (d) Leviathan

VI. Conclusion

Episode rules are often used for next event sequence prediction as they are not only accurate but also easily interpreted by humans. In this study, we improved upon this approach by utilizing a new type of episode rules, named partially ordered episode rules. These rules are discovered by loosening the ordering constraints between events in each rule's antecedent and consequent. Extensive experiments on four datasets have shown that this can considerably reduce the number of rules and provide higher accuracy compared to traditional episode rules and a recent type of rules called the precise-positioning episode rules.

In future work, episode rule-based sequence prediction models will be developed to handle more complex event sequences. In particular, the usage of a taxonomy [11] and the addition of various utility functions [12]–[16] will be considered. Besides, an incremental episode rule mining algorithm will be designed to decrease runtime performance in dynamic environment such as a stream [17].

References

[1] C. C. Aggarwal, *Data Mining: The Textbook*. Cham: Springer, 2015.

[2] H. Amirat, N. Lagraa, P. Fournier-Viger, and Y. Ouinten, "Nextroute: a lossless model for accurate mobility prediction," *J. Ambient Intell. Humaniz. Comput.*, vol. 11, no. 7, pp. 2661–2681, 2020.

[3] M. Nouioua, P. Fournier-Viger, G. He, F. Nouioua, and Z. Min, "A survey of machine learning for network fault management," in *Machine Learning and Data Mining for Emerging Trends in Cyber Dynamics*. Springer, 2021, pp. 1–27.

[4] X. Ao, P. Luo, J. Wang, F. Zhuang, and Q. He, "Mining precise-positioning episode rules from event sequences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 3, pp. 530–543, 2017.

[5] K. Huang and C. Chang, "Efficient mining of frequent episodes from complex sequences," *Inf. Syst.*, vol. 33, no. 1, pp. 96–114, 2008.

[6] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in sequences," in *Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining*, 1995.

[7] P. Fournier-Viger, Y. Chen, F. Nouioua, and J. C. Lin, "Mining partially-ordered episode rules in an event sequence," in *Proc. 13th Asian Conference on Intelligent Information and Database Systems*. Springer, 2021, pp. 3–15.

[8] P. Fournier-Viger, Y. Yang, P. Yang, J. C.-W. Lin, and U. Yun, "Tke: Mining top-k frequent episodes," in *Proc. 33rd Intern. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2020.

[9] K. Iwanuma, Y. Takano, and H. Nabeshima, "On anti-monotone frequency measures for extracting sequential patterns from a single very-large data sequence," in *Proc. 1st International Workshop on Knowledge Discovery in Data Streams, in conjunction with ECML/PKDD*, 2004.

[10] P. Fournier-Viger, J. C.-W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The spmf open-source data mining library version 2," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2016, pp. 36–40.

[11] X. Ao, H. Shi, J. Wang, L. Zuo, H. Li, and Q. He, "Large-scale frequent episode mining from complex event sequences with hierarchies," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 4, pp. 1–26, 2019.

[12] Y.-F. Lin, C.-W. Wu, C.-F. Huang, and V. S. Tseng, "Discovering utility-based episode rules in complex event sequences," *Expert Systems with Applications*, vol. 42, no. 12, pp. 5303–5314, 2015.

[13] S. Rathore, S. Dawar, V. Goyal, and D. Patel, "Top-k high utility episode mining from a complex event sequence," in *Proc. 21st Int. Conf. on Management of Data*, 2016, pp. 56–63.

[14] W. Song and C. Huang, "Mining high average-utility itemsets based on particle swarm optimization," *Data Science and Pattern Recognition*, vol. 4, no. 2, pp. 19–32, 2020.

[15] C. Wu, Y. Lin, P. S. Yu, and V. S. Tseng, "Mining high utility episodes in complex event sequences," in *Proc. 19th ACM SIGKDD Int. Conf. on Knowl. Discovery*, 2013, pp. 536–544.

[16] U. Yun, H. Nam, G. Lee, and E. Yoon, "Efficient approach for incremental high utility pattern mining with indexed list structure," *Future Generation Computer Systems*, vol. 95, pp. 221–239, 2019.

[17] T. You, Y. Li, B. Sun, and C. Du, "Multi-source data stream online frequent episode mining," *IEEE Access*, vol. 8, pp. 107 465–107 478, 2020.