MaxFEM: Mining Maximal Frequent Episodes in Complex Event Sequences

Philippe Fournier-Viger[1][0000−0002−7680−9899], M. Saqib Nawaz[1][0000−0001−9856−2885], Yulin He[1][0000−0002−3415−0686] Youxi Wu[2][0000−0001−5314−3468], Farid Nouioua[0000−0002−6505−9797][2], and Unil Yun[4][0000−0002−3720−0861]

[1] Shenzhen University, Shenzhen, China
[2] Hebei University of Technology, Tianjin, China
[3] University of Bordj Bou Arreridj, Algeria
[4] Department of Computer Engineering, Sejong University, Seoul, Republic of Korea.
{philfv, msaqibnawaz}@szu.edu.cn, wuc567@163.com, faridnouioua@gmail.com, yunei@sejong.ac.kr

Abstract. Frequent episode mining is a key data mining task, for analyzing discrete sequences, used in many domains. The goal is to enumerate all subsequences of symbols or events that are appearing at least some minimum number of times. In the last decades, several efficient episode mining algorithms were designed. Nonetheless, a major issue is that they often yield a huge number of frequent episodes, which is inconvenient for users. As a solution, this paper presents an efficient algorithm called MaxFEM (Maximal Frequent Episode Miner) to identify only the maximal frequent episodes in a complex sequence. The key benefit is that the number of frequent episodes presented to the user can be reduced by several times. The MaxFEM algorithm includes many strategies to improve its performance. An experimental evaluation on benchmark datasets confirms that MaxFEM has excellent performance.

Keywords: Pattern Mining · Frequent Episodes · Maximal Episodes.

1 Introduction

Analyzing data to discover frequent patterns is a popular research topic in data mining. Over the last decades, various algorithms have been designed to analyze various types of data such as transactions, sequences, graphs and trees. Among those data types, more and more attention is given to discrete sequences. A *discrete sequence* is an ordered list of events or symbols and can be used to encode varied data such as sequences of moves in a Chess game, sequences of clicks on a website, sequences of alarms generated by a system [13], sequences of nucleotides in a virus genome [14], and sequences of words in a novel [5].

To find interesting patterns in sequences, a large body of research has focused on designing algorithms to extract frequent subsequences in discrete sequences. These studies can be generally categorized as addressing one of two tasks: *sequential pattern mining* (SPM)[5, 15] and *frequent episode mining* (FEM) [13].

The former task consists of finding patterns that are common to a set of discrete sequences, while the latter aims at finding patterns in a very long sequence. Algorithms for these problems are quite different and both problems have many real-life applications. In this paper, the focus is on FEM.

The input to FEM is a discrete sequence, a window size $maxWindow$ and a threshold called the minimum support ($minsup$). The output is all the frequent episodes, that is the subsequences that appears at least $minsup$ times in the input sequence. Finding frequent episodes is useful but very difficult computationally, especially for long sequences, low $minsup$ values and large $maxWindow$ values. For this reason, several efficient algorithms have been proposed. The first two algorithms are MINEPI and WINEPI [13]. WINEPI can identify serial episodes (where events are all sequentially ordered), parallel episodes (where all events are unordered) and composite episodes (where events are partially ordered). To find frequent episodes, WINEPI employs a breadth-first search and utilizes a sliding-window model. WINEPI counts the occurrence frequency (also called support) of an episode as the number of windows that contains an occurrence of the episode. As noted by Iwanuma et al. [9], a drawback of that definition is that a same occurrence may be counted multiple times. To avoid this problem, MINEPI was designed. It is a breadth-first search algorithm that only counts the minimal occurrences of each episode [13]. Thereafter, another occurrence counting function was proposed called the *head frequency*, which is used by several recent FEM algorithms such as EMMA, MINEPI+[8], and TKE [7] as it is more suitable for prediction [8]. EMMA and TKE rely on a depth-first search in combination with a memory anchor technique to speed up the search, and were shown to outperform several earlier algorithms such as MINEPI [13] and MINEPI+ [8] by a large margin. Research on FEM is ongoing and new algorithms and extensions are published regularly such as for discovering extended episode types such as high utility episodes [6, 12] and online episodes [2].

Though frequent episodes can reveal useful information, a major issue is that current FEM algorithms can generate huge result sets, sometimes containing millions of frequent episodes, and that these episodes are often very similar to each other. For instance, when analyzing the data of a customer in a store, a frequent episode may indicate that the person bought milk, then bread, and then some oranges. But all the subsequences of this pattern would generally be also frequent such as the episode of buying milk followed by bread, the episode of buying bread followed by oranges, or the episode of buying milk followed by oranges. This is a major problem because all these patterns can be viewed as redundant as they are included in the first episode, and combing through large sets of episodes can be very time-consuming for users.

In recent years, some researchers have attempted to propose a solution to this problem by designing algorithms to discover concise representations of frequent episodes such as closed episodes [1, 11] and maximal episodes [3]. The aim is to find a subset of all episodes that summarize them. But the majority of these algorithms are only able to analyze *simple sequences* (without simultaneous events) [1, 3, 11]. This greatly simplifies the problem of mining episodes but

makes these algorithms unusable for analyzing many real life event sequences such as customer transactions (as customers may buy multiple products at the same time). Thus, it is important to address the general problem of mining concise representations of episodes in complex sequences (with simultaneous events).

To address this issue, this paper proposes a novel algorithm called MaxFEM (Maximal Frequent Episode Miner) to mine maximal frequent episodes in complex event sequences. A maximal frequent episode is a frequent episode that is not included in larger frequent episodes. The key benefit of mining maximal episodes is that the number of frequent episodes presented to the user can be greatly reduced, as it will be shown in the experimental evaluation. To our best knowledge, MaxFEM is the first algorithm to discover maximal episodes in complex sequences. To efficiently discover the maximal frequent episodes, MaxFEM includes three strategies to improve its performance, named Efficient Filtering of Non-maximal episodes (EFE), Skip Extension checking (SEC), and Temporal pruning (TP). An experimental evaluation on several benchmark datasets confirms that MaxFEM has excellent performance.

The structure of the rest of this paper is the following. Section 2 describes the problem of FEM and the novel problem of maximal FEM. Then, Section 3 presents the MaxFEM algorithm. Section 4 reports results for the experimental evaluation. Lastly, Section 5 draws a conclusion and list several opportunities for future work.

2 Problem Definition

This section gives a formal definition of the problem of frequent episode mining, discusses its properties, and then describes the proposed problem of maximal frequent episode mining.

The input data in frequent episode mining is a discrete sequence [8, 13]. Assume that there is a finite set $E = \{i_1, i_2, \ldots, i_m\}$ of *events* (also called items or symbols). A subset $X \subseteq E$ is said to be *an event set*. A *discrete sequence*, also called a *complex event sequence*, is defined as a finite ordered list $S = \langle (SE_{t_1}, t_1), (SE_{t_2}, t_2), \ldots, (SE_{t_n}, t_n) \rangle$ of pairs of the form (SE_{t_i}, t_i) where $SE_{t_i} \in E$ is an event set and t_i is an integer representing a timestamp. A sequence is ordered by time, that is for any integers $1 \leq i < j \leq n$, the relationship $t_i < t_j$ holds. An event set SE_{t_i} of a sequence contains events that are assumed to have occurred at the same time, and for this reason it is called a *simultaneous event set*. In the case, where a complex event sequence contains event sets each only having one event, it is said to be a *simple event sequence*. It is to be noted that a same event can appear multiple times in a sequence (in different event sets). Besides, although the definition of sequence includes timestamps, it can also be used to model sequences that do not have timestamps such as sequence of words by assigning contiguous integers as timestamps (e.g. $1, 2, 3, 4, 5$).

To illustrate these definitions, a complex event sequence is presented in Fig. 1, which has eight event sets and timestamps ranging from 1 to 11. This sequence is formally represented as $S = \langle (\{a, c\}, 1), (\{a\}, 2), (\{a, b\}, 3), (\{a\}, 6), (\{a, b\}, 7),$

$(\{c\}, 8)$, $(\{b\}, 9)$, $(\{d\}, 11)\rangle$, and will be used as running example. The interpretation of this sequence is that events a and c occurred at time 1, were followed by event a at time 2, and then by a and b at time 3. Then, the event a was observed at time 6, the events a and b at time 7, the event c at time 8, the event b at time 9, and lastly event d at time 11. As shown in this example, timestamps are not required to be contiguous. This type of sequences can encode various data such as sequence of events from a complex system, network data [10], cloud data [1], malicious attacks [16], and stock data [12].

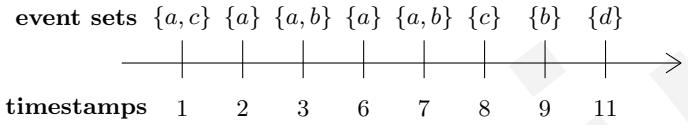

Fig. 1: A complex event sequences with eight event sets

The problem of *frequent episode mining* is to find all frequent episodes in a complex event sequence. A frequent episode is an episode that has a high support (has many occurrences in the sequence). There are three types of episodes [13]. A *composite episode* α is an ordered list of simultaneous event sets. A composite episode α having p event sets is represented as $\alpha = \langle X_1, X_2, \ldots, X_p\rangle$, where $X_i \subseteq E$, and X_i is said to appear before X_j for any integers $1 \leq i < j \leq p$. The *size* of α is defined as $size(\alpha) = \bigcup_{i\in[1,p]} |X_i|$. A *parallel episode* is a composite episode that contains a single event set. A *serial episode* is a composite episode where each event set contains one event. Several FEM mining algorithms are only able to handle the special case of serial episodes, while others can find all composite episodes [13, 8].

To find frequent episodes, the concept of support is crucial (how to count the occurrences in a sequence). Multiple support functions have been proposed, which have different advantages and limitations. In this study, the *head frequency* support function [9] is used, which has been used in several algorithms such as MINEPI+ [8], EMMA [8] and TKE [7]. The concept of occurrence is first presented and then the head support function.

Definition 1 (Occurrence). *An* occurrence *of an episode $\alpha = \langle X_1, X_2, \ldots, X_p\rangle$ in a complex event sequence $S = \langle (SE_{t_1}, t_1), (SE_{t_2}, t_2), \ldots, (SE_{t_n}, t_n)\rangle$ is a time interval $[t_s, t_e]$ that satisfies $X_1 \subseteq SE_{z_1}$, $X_2 \subseteq SE_{z_2}$, ..., $X_p \subseteq SE_{z_w}$ for some integers $t_s = z_1 < z_2 < \ldots < z_w = t_e$. In an occurrence $[t_s, t_e]$, t_s is said to be the start point, while t_e is the end point. The length of an occurrence $[t_s, t_e]$ is defined as $t_s - t_e$. The notation $occSet(\alpha)$ refers to the* set of all occurrences *of α that have a length that is smaller than some maximum length winlen set by the user.*

As example, if $winlen = 3$, the composite episode $\alpha = \langle\{a\}, \{a, b\}\}\rangle$ has an occurrence set with three occurrences, i.e. $occSet(\alpha) = \{[t_1, t_3], [t_2, t_3], [t_6, t_7]\}$.

Definition 2 (Head support). *Let there be a composite sequence S and an episode α. The support of α in S is defined as $sup(\alpha) = |\{t_s | [t_s, t_e] \in occSet(\alpha)\}|$, that is the number of distinct start points in the occurrence set of α* [8].

Continuing the previous example, the support of $\alpha = \langle\{a\}, \{a, b\}\rangle$ is $sup(\alpha) = 3$ because there are three different start points in $occSet(\alpha)$, namely t_1, t_2, and t_6.

Based on the above definitions, the problem of mining frequent episodes is defined as:

Definition 3 (Mining frequent episodes). *Given, a complex event sequence S, a user-defined threshold $minsup > 0$ and a user-specified window length $winlen > 0$, the problem of mining frequent episodes is to enumerate all frequent episodes appearing in S. An episode α is frequent if $sup(\alpha) \geq minsup$* [8].

For instance, for $minsup = 2$ and $winlen = 3$, there are seven frequent episodes in the sequence depicted in Fig. 1. Those are $\langle\{a\}\rangle$, $\langle\{b\}\rangle$, $\langle\{c\}\rangle$ $\langle\{a\}, \{b\}\rangle$, $\langle\{a, b\}\rangle$, $\langle\{a\}, \{a\}\rangle$, and $\langle\{a\}, \{a, b\}\rangle$. The support of these episodes are respectively 5, 3, 2, 2, 2, 3, and 2.

Several algorithms were proposed to solve the problem of frequent episode mining. To find the frequent episodes without considering all possible episodes, these algorithms use a powerful search space pruning property of the support, called the downward closure property, which indicates that the support of an episode cannot be greater than that of its prefix episodes [8]. Formally, this means that the relationship $sup(\alpha) \leq sup(\beta)$ holds for any episode $\beta = \langle X_1, X_2, \ldots, X_i\rangle$ and episode $\alpha = \langle X_1, X_2, \ldots, X_p\rangle$ where $i < p$.

A major problem with current algorithms for FEM is that too many frequent episodes may be discovered. To address this issue, this paper proposes to discover only the maximal frequent episodes. This problem is defined as follows.

Definition 4 (Mining maximal frequent episodes in a complex event sequence). *Given, a complex event sequence S, a user-defined threshold $minsup > 0$ and a user-specified window length $winlen > 0$, the problem of mining maximal frequent episodes is to enumerate all frequent episodes that are not strictly included in another frequent episode* [8]. *An episode $\alpha = \langle Y_1, Y_2, \ldots, Y_i\rangle$ is strictly included in an episode $\beta = \langle X_1, X_2, \ldots, X_p\rangle$ if and only if $Y_1 \subseteq X_{k_1}$, $Y_2 \subseteq X_{k_2} \ldots Y_i \subseteq X_{k_i}$ for some integers $1 \leq k_1 < k_2 < \ldots < k_i \leq p$. This relation is denoted as $\alpha \sqsubseteq \beta$.*

For instance, in the same example, there are only two maximal frequent episodes that are $\langle\{c\}\rangle$ and $\langle\{a\}, \{a, b\}\rangle$. Thus, five non-maximal frequent episodes are omitted, which can be viewed as redundant, as they are strictly included in the maximal episodes.

A naive approach to solve the problem of mining maximal frequent episodes would be to first discover all frequent episodes using a traditional algorithm such as EMMA [8] or TKE [7], and then to perform a post-processing step to compare frequent episodes to filter episodes that are non-maximal. This approach

would work. However, it is inefficient because it requires keeping in memory all frequent episodes, and there can be a huge number. As it will be shown in the experimental evaluation, the number of maximal episodes can be much smaller than the number of frequent episodes. Hence, it is desirable to design an algorithm that does not require maintaining all frequent episodes in memory. The next section presents the MaxFEM algorithm.

3 The MaxFEM algorithm

The MaxFEM algorithm is the first algorithm for mining maximal frequent episodes in a complex event sequence (the general case). Also, differently from prior work [3], MaxFEM relies on the head frequency to count the support of episodes [7, 8]. To explore the search space of frequent episodes, MaxFEM performs a depth-first search using the basic search procedure of EMMA [8]. This procedure was selected as basis for MaxFEM because it is efficient for exploring the search space of frequent episodes with the head frequency. The procedure consists of first discovering frequent single events, then to combine these events to find frequent parallel episodes, and finally, to join the frequent parallel episodes to obtain frequent composite episodes. However, this procedure is not designed for identifying the maximal frequent episodes.

To find only the maximal frequent episodes and avoid the naive solution of filtering non-maximal episodes by post-processing, MaxFEM adopts the following approach. MaxFEM is equipped with a data structure called W for storing at any moment the current maximal frequent episodes. Then, MaxFEM starts searching for episodes. When MaxFEM finds a new frequent composite episode X, MaxFEM compares X with the episodes already in W. If X is strictly included in an episode already in W, then X is ignored as it is not maximal. In the other case where X is maximal, any episode in W that is strictly included in X is removed from W. When the algorithm terminates, this ensure obtaining the maximal frequent episodes.

The next paragraphs present each step of the MaxFEM algorithm. Then, three additional optimizations are introduced to obtain a more efficient algorithm.

The MaxFEM algorithm takes as input a complex event sequence S, and the *minsup* and *winlen* parameters. MaxFEM outputs the maximal frequent episodes. The pseudocode is shown in Algorithm 1. The key steps are the following:

Step 1. Finding the frequent events. MaxFEM reads the input sequence S to compute the support of each single event, to then identify the set E' of all frequent events.

For instance, consider the input sequence $S = \langle(\{a, c\}, 1), (\{a\}, 2), (\{a, b\}, 3), (\{a\}, 6), (\{a, b\}, 7), (\{c\}, 8), (\{b\}, 9), (\{d\}, 11)\rangle$ depicted in Fig. 1, $minsup = 2$ and $winlen = 3$. The frequent events are a, b and c, since they have a support of 5, 3 and 2, respectively, which is no less than *minsup*. The event d is infrequent because it has a support of $1 < minsup$.

Algorithm 1: The MaxFEM algorithm

input : S: an input sequence, *minsup*: a user-specified threshold, *winlen*: the window length
output: the maximal frequent episodes

Scan S to calculate $sup(e)$ for each event $e \in E$;
$E' \leftarrow \{e | e \in E \wedge sup(e) \geq minsup\}$;
Thereafter, ignore or remove each event $e \notin E'$ from S;
Read S to build the location list of each frequent event $e \in E'$;
$PEpisodes \leftarrow E'$;
foreach *parallel episode* $ep \in PEpisodes$ *such that* $sup(ep) \geq minup$ **do**
  foreach *event* $e \in E'$ *such that* $sup(e) \geq minup$ **do**
    $newE \leftarrow parallelExtension(ep, e)$;// and build $newE$'s location list
    if $sup(newE) \geq minsup$ **then**
      $PEpisodes \leftarrow PEpisodes \cup \{newE\}$;
    end
  end
end
$W \leftarrow PEpisodes$;
Re-encode the sequence S into a sequence S' using the parallel episodes;
foreach *composite episode* $ep \in W$ *such that* $sup(ep) \geq minup$ **do**
  foreach *event* $e \in PEpisodes$ *such that* $sup(e) \geq minup$ **do**
    $newE \leftarrow sExtension(ep, e)$;// and build $newE$'s bound list
    if $sup(newE) \geq minsup$ *and* $newE$ *has no superset in* W **then**
      $W \leftarrow W \cup \{newE\}$;
      Remove all subsets of $newE$ that are in W;
    end
  end
end
Return W;

Thereafter, infrequent events will be ignored as they cannot appear in frequent episodes.

Step 2. Build the location-lists of frequent events. Next, MaxFEM reads the input sequence S again to create a vertical structure, named *location list* [8] for each frequent event. A formal definition of this structure is given:

Definition 5 (Location list). *Let there be an input sequence* $S = \langle (SE_{t_1}, t_1), (SE_{t_2}, t_2), \ldots, (SE_{t_n}, t_n) \rangle$. *Furthermore, assume that events from each event set in* S *are sorted by a total order* $\prec$ *on events. This order can be any order such as the lexicographical order* ($a \prec b \prec c \prec d$). *If an event* e *is included in the* i*-th event set* SE_{t_i} *of the input sequence, then* e *is said to appear at* position $\sum_{w=1,\ldots,i-1} |SE_{t_w}| + |\{y | y \in SE_{t_i} \wedge y \prec e\}|$. *For an event* e, *its* location list *in the sequence* S *is the list of its timestamps and is denoted as* $locList(e)$. *An interesting property is that the support of an event* e *can be obtained from its location list as* $sup(e) = |locList(e)|$.

Continuing the running example, the location lists of frequent events a, b and c are $locList(a) = \{1, 2, 3, 6, 7\}$, $locList(b) = \{3, 7, 9\}$ and $locList(c) = \{1, 8\}$, respectively.

Step 3. Finding the frequent parallel episodes. In the third step, frequent episodes are recursively extended to find all frequent parallel episodes, as in EMMA [8]. A set to store frequent parallel episodes, called $PEpisodes$, is created and initialized as $PEpisodes = E'$. Then, MaxFEM tries to extend each frequent episode $ep \in PEpisodes$ by combining it with each frequent event $e \in E' | e \notin ep \wedge \forall f \in ep, f \prec e$ to obtain a larger parallel episode $newE = ep \cup \{e\}$. The resulting episode $newE$ is said to be a *parallel extension* of ep with event e. To determine the support of $newE$, its location list is built by intersecting the location lists of e and ep. That is the location list of $newE$ is created as $locList(newE) = locList(e) \cap locList(ep)$. If the support $|locList(newE)|$ is equal or greater than $minsup$, $newE$ is added to $PEpisodes$ (with its location list) because it is a frequent parallel episode. After recursively performing parallel extensions of episodes in $PEpisodes$, this latter contains all frequent parallel episodes. For instance, the episode $\langle\{a\}\rangle$ can be extended with the frequent event c, to obtain the parallel episode $\langle\{a, c\}\rangle$. The location list of that episode is $loclist(\langle\{a, c\}\rangle) = loclist(\langle\{a\}\rangle) \cap loclist(\langle\{c\}\rangle) = \{1, 2, 3, 6, 7\} \cap \{1, 8\} = \{1\}$. Hence, the support of $\langle\{a, c\}\rangle$ is $sup(\langle\{a, c\}\rangle) = |loclist(\langle\{a, c\}\rangle)| = 1$ and this parallel extension is infrequent. After repeating this process to generate all parallel extensions, it is found that parallel frequent episodes are: $\langle\{a\}\rangle$, $\langle\{b\}\rangle$, $\langle\{c\}\rangle$, and $\langle\{a, b\}\rangle$. Their support values are 5, 3, 2, 2, respectively.

Step 4. Using parallel episodes to re-encode the sequence. Next, MaxFEM assigns a unique identifier to each parallel frequent episode. Then, the algorithm transforms the input sequence S into a *re-encoded sequence* S' by replacing events from S by parallel episodes. For the running example, MaxFEM assigns #1, #2, #3 and #4 as identifiers for the episodes $\langle\{a\}\rangle$, $\langle\{b\}\rangle$, $\langle\{c\}\rangle$, and $\langle\{a, b\}\rangle$. Then, S is re-encoded as: $S' = \langle(\{\#1\#3\}, 1), (\{\#1\}, 2), (\{\#1, \#2, \#4\}, 3), (\{\#1\}, 6), (\{\#1, \#2, \#4\}, 7), (\{\#3\}, 8), (\{\#2\}, 9)\rangle$.

Step 4. Finding the maximal frequent composite episodes. Thereafter, MaxFEM searches for frequent maximal composite episodes using the re-encoded sequence S'. A data structure W is first initialized to store the maximal frequent composite episodes. All parallel frequent episodes are added to W, as they are currently considered to be maximal. Then, MaxFEM attempts to build larger frequent composite episodes by recursively performing serial extensions of episodes in W. A serial extension is the combination of an episode $ep = \langle SE_1, SE_2, \ldots, SE_x\rangle \in W$ with a parallel episode $pe \in PEpisodes$ to obtain a larger composite episode $sExtension(ep, pe) = \langle SE_1, SE_2, \ldots, SE_x, pe\rangle$.

For each serial extension $sExtension(ep, pe)$, MaxFEM creates its *bound list* structure, which is defined as:

Definition 6 (Bound list). *Let there be a re-encoded sequence* $S' = \langle(SE_{t_1}, t_1), (SE_{t_2}, t_2), \ldots, (SE_{t_n}, t_n)\rangle$. *The bound list of a parallel episode* pe *is defined as* $boundList(pe) = \{[t, t] | pe \subseteq SE_t \in S'\}$. *The bound list of the serial extension of a composite episode* ep *with* pe, *is defined as:* $boundList(sExtension(ep, pe)) =$

$\{[u,w]|\ [u,v] \in boundList(ep) \wedge [w,w] \in boundList(pe) \wedge w - u < winlen \wedge v < w\}$. *The bound list of a composite episode ep allows deriving its support as* $sup(ep) = |\{t_s|[ts,te] \in boundList(ep)\}|$.

MaxFEM combines each episode in W with each parallel episode appearing in a same window *winlen* in S' to create serial extensions. If an extension $sExtension(ep, pe)$ is frequent and not strictly included in an episode already in W, then (1) it is added to W and (2) each episode $ee \in W$ that is strictly included in $sExtension(ep, pe)$ is removed from W because it is not maximal. This process ensures maintaining the current maximal frequent composite episodes in W at any moment. When no more serial extensions can be done, W contains all maximal frequent episodes and W is returned to the user.

As example, consider the serial extension of $\langle\{a\}\rangle$ with $\langle\{a\}\rangle$, which results in $f = \langle\{a\},\{a\}\rangle$. The bound list of f is $boundList(f) = \{[t_1,t_2],\ [t_2,t_3],\ [t_6,t_7]\}$. Hence, $sup(f) = |\{t_1,t_2,t_6\}| = 3$. Since this serial extension is frequent, it is added to W and $\langle\{a\}\rangle$ is removed from W. This process is repeated for other serial extensions. In the end, the set of maximal frequent episodes W is: $\langle\{c\},\{a\}\rangle$, $\langle\{a\},\{c\}\rangle$ and $\langle\{a\},\{a,c\}\rangle$, with a support of 2, 2, and 3 respectively (the end result).

Completeness. It can be seen that MaxFEM is a complete algorithm as it relies on the search procedure of EMMA to explore the search space of frequent episodes, and MaxFEM only eliminates non-maximal episodes during the final step where composite episodes are generated (Step 4).

It can be tempting to also eliminate non-maximal episodes during the earlier step of generating parallel episodes (Step 3). But if this would be done, the algorithm would become incomplete. This is demonstrated by an example. If the parallel episode $\langle\{a\}\rangle$ is eliminated early in Step 3 because it is strictly included in the parallel episode $\langle\{a,c\}\rangle$, then the maximal episode $\langle\{a\},\{a,c\}\rangle$ will not be generated in Step 4 and thus it would be missed in the final result.

Optimizations. MaxFEM applies three strategies to improve performance.

Strategy 1. Efficient Filtering of Non-maximal episodes (EFE). The first strategy consists of using an appropriate data structure to implement W and to optimize the two operations that are done using it: (1) searching for episodes strictly included in an episode e (sub-episode checking) and (2) searching for episodes in which e is strictly included (super-episode checking). Because these checks are relatively costly, two ideas are used to reduce the number of checks.

First, W is implemented as a list of heaps $W = \{W_1, W_2, ...W_n\}$ where n is the size of the longest frequent episode discovered until now. In W, the heap W_x $(1 \leq x \leq n)$ stores the maximal episodes found until now of size x. Then, to do super-episode (sub-episode) checking for an episode ep having w events, MaxFEM only compares ep with episodes in $W_{w+1}, W_{w+2}...W_n$ $(W_1, W_2...W_{w-1})$. This is because an episode can only be strictly included (strictly include) an episode if it has a larger (smaller) size.

Second, each event from the input sequence is encoded as a distinct integer. Then, a hash value $hash(ep)$ is calculated for each episode ep as the sum of its events. For instance, if the events a, b, c are encoded as 1, 2, and 3, the

hash value of the episode $\langle\{a\},\{a,c\}\rangle$ is $1 + 1 + 3 = 5$ Based on these hash values, episodes stored in each heap of W are sorted by decreasing hash values. This allows optimizing super-episode checking as follows. For a heap W_x and an episode α, if $hash(\alpha) > hash(\beta)$ for any episode $\beta \in W_x$, then it is unnecessary to check if $\alpha \sqsubset \beta$ for β and all episodes after β in W_x. Similarly, for a heap W_x and an episode α, if $hash(\beta) > hash(\alpha)$ for any episode $\beta \in W_x$, then it is unnecessary to check $\beta \sqsubset \alpha$ as well as all episodes after β in W_x when W_x is traversed in reverse order.

Strategy 2. Skip Extension checking (SEC). This strategy is based on the depth-first exploration of composite episodes. If a frequent episode *ep* is extended by serial extension to form another frequent episode, then it is unnecessary to do super-pattern and sub-pattern checking for *ep* as *ep* cannot be maximal. Thus, *ep* is only considered to be added to W if it has no frequent serial extensions.

Strategy 3. Temporal pruning (TP). The third optimization aims at reducing the cost of creating bound lists. Creating the bound list of an extension $sExtension(ep, pe)$ requires to compares elements in the bound lists of *ep* and *pe* one by one. If at any point the number of remaining elements is not enough to satisfy *minsup*, the construction of the bound-list is stopped.

4 Experimental Evaluation

To assess the performance of the designed MaxFEM algorithm, a set of experiments have been carried out. The runtime of MaxFEM was compared with the EMMA [8] algorithm. EMMA is selected as baseline as MaxFEM relies on the search procedure of EMMA, and they both use the head frequency measure for counting episode occurrences, and they mine composite episodes for the general case of a complex sequence. Also, EMMA is also faster than some recent algorithms such as TKE [7]. All algorithms have been implemented in Java and the experiments were run on a laptop with Windows 11 and a Intel Core i7-8565U CPU @ 1.80 GHz and 16 GB of RAM. The memory usage of algorithms was captured using the Java API. Datasets and source code of the algorithms are available in the SPMF library (`www.philippe-fournier-viger.com/spmf`) [4].

Several datasets have been used and gave similar results but due to space limitations results for only two datasets are shown, called *Kosarak* and *Retail*, which are popular benchmark datasets for pattern mining and represent different data types. *Kosarak* is click-stream dataset from a Hungarian news portal containing 990,000 event sets, 41,270 distinct event types and an average event set size of 8.1 items. *Retail* is transaction data from a Belgian retail store containing 88,162 event sets, 16,470 distinct event types and an average event set size of 10.3 items. The characteristics of the datasets are presented in Table 1.

Algorithms were run on each dataset with $winlen \in \{5, 10, 15\}$ while *minsup* was decreased until a clear performance trend was observed or algorithms would fail to terminate due to a 300 seconds time limit set for experiments. Results shown in Fig. 2 compares the runtime and number of patterns found by each algorithm. It is observed that MaxFEM is always about 10% to 40% faster than

Table 1: Characteristics of the datasets

Dataset	Avg. Sequ. Len.	#Events	#Sequences	Density(%)
Kosarak	8.1	41,270	990,000	0.02
Retail	10.3	16,470	88,162	0.06

EMMA. This is due to the three novel optimizations since EMMA and MaxFEM uses the same basic search procedure.

It is also observed that the number of maximal episodes is much smaller than all frequent episodes. For example, on Kosarak for $minsup = 20,000$ and $winlen = 5$, MaxFEM finds 694 maximal episodes, while EMMA finds 2,583 frequent episodes. Thus, it can be concluded that the performance of MaxFEM is acceptable and maximal episodes provide a compact summary of all frequent episodes. Results (not shown) on other tested datasets are similar.

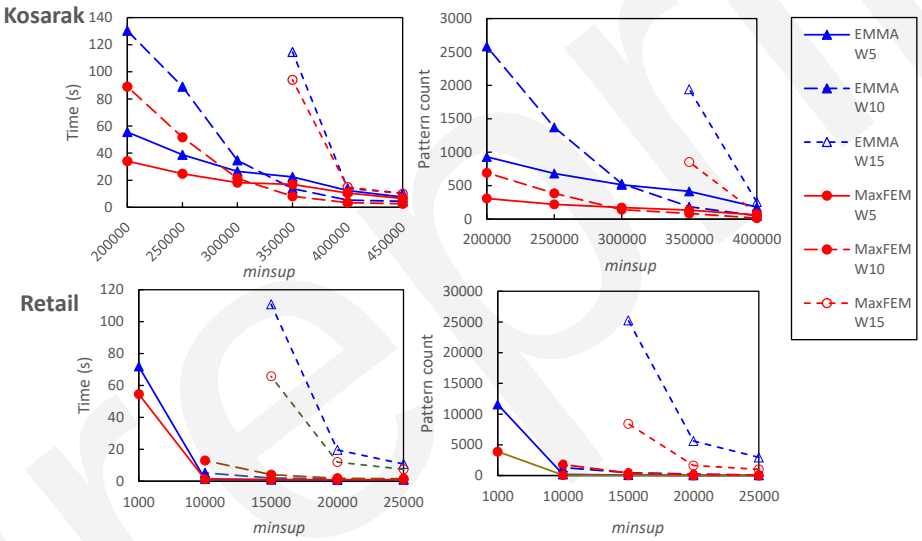

Fig. 2: Comparison of runtime and pattern count

5 Conclusion

This paper has proposed a novel algorithm named MaxFEM for discovering maximal episodes for the general case of a complex event sequence, and using the head frequency function. MaxFEM includes three strategies to improve its performance, named Efficient Filtering of Non-maximal episodes (EFE), Skip Extension checking (SEC), and Temporal pruning (TP). An experimental evaluation on real datasets has shown that maximal episodes provides a compact summary of all frequent episodes and that MaxFEM has a significant speed advantage over the EMMA algorithm. In future work, an interesting plan is

to extend MaxFEM for other frequency functions and sequences types and to design a parallel and distributed version.

The source code of MaxFEM is available in the SPMF library [4], as well as a version of MaxFEM for mining all frequent episodes called AFEM (All Frequent Episode Miner).

References

1. Amiri, M., Mohammad-Khanli, L., Mirandola, R.: An online learning model based on episode mining for workload prediction in cloud. Future Generation Computer Systems **87**, 83–101 (2018)
2. Ao, X., Luo, P., Li, C., Zhuang, F., He, Q.: Online frequent episode mining. In: Proc. 31st IEEE Int. Conf. on Data Eng. pp. 891–902 (2015)
3. Ao, X., Shi, H., Wang, J., Zuo, L., Li, H., He, Q.: Large-scale frequent episode mining from complex event sequences with hierarchies. ACM Transactions on Intelligent Systems and Technology **10**(4), 1–26 (2019)
4. Fournier-Viger, P., Lin, J.C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H.T.: The SPMF open-source data mining library version 2. In: Proc. 20th European conference on machine learning and knowledge discovery in databases. pp. 36–40. Springer (2016)
5. Fournier-Viger, P., Lin, J.C.W., Kiran, U.R., Koh, Y.S.: A survey of sequential pattern mining. Data Science and Pattern Recognition **1**(1), 54–77 (2017)
6. Fournier-Viger, P., Yang, P., Lin, J.C.W., Yun, U.: HUE-Span: Fast high utility episode mining. In: Proc. 14th Intern. Conf. on Advanced Data Mining and Applications. pp. 169–184. Springer (2019)
7. Fournier-Viger, P., Yang, Y., Yang, P., Lin, J.C.W., Yun, U.: TKE: Mining top-k frequent episodes. In: Proc. 33rd Intern. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems. Springer (2020)
8. Huang, K., Chang, C.: Efficient mining of frequent episodes from complex sequences. Inf. Syst. **33**(1), 96–114 (2008)
9. Iwanuma, K., Takano, Y., Nabeshima, H.: On anti-monotone frequency measures for extracting sequential patterns from a single very-long data sequence. In: Proc. IEEE Conf. on Cybernetics and Intelligent Systems. vol. 1, pp. 213–217 (2004)
10. Li, L., Li, X., Lu, Z., Lloret, J., Song, H.: Sequential behavior pattern discovery with frequent episode mining and wireless sensor network. IEEE Communications Magazine **55**(6), 205–211 (2017)
11. Liao, G., Yang, X., Xie, S., Yu, P.S., Wan, C.: Mining weighted frequent closed episodes over multiple sequences. Tehnički vjesnik **25**(2), 510–518 (2018)
12. Lin, Y., Huang, C., Tseng, V.S.: A novel methodology for stock investment using high utility episode mining and genetic algorithm. Appl. Soft Comput. **59**, 303–315 (2017)
13. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovering frequent episodes in sequences. In: Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining (1995)
14. Nawaz, M.S., Fournier-Viger, P., Shojaee, A., Fujita, H.: Using artificial intelligence techniques for COVID-19 genome analysis. Appl. Intell. **51**(5), 3086–3103 (2021)
15. Nawaz, M.S., Sun, M., Fournier-Viger, P.: Proof guidance in PVS with sequential pattern mining. In: Proc. of 8th Intern. Conf. on Fundamentals of Software Engineering. pp. 45–60. Springer (2019)
16. Su, M.Y.: Applying episode mining and pruning to identify malicious online attacks. Computers & Electrical Engineering **59**, 180–188 (2017)