IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 23, 2019, accepted January 7, 2020, date of publication January 10, 2020, date of current version January 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2965647

Discovery of Frequent Patterns of Episodes Within a Time Window for Alarm Management Systems

ADEL HIDRI[1], **AHMED SELMI**[2], **AND MINYAR SASSI HIDRI**[1]
[1]Computer Department, Deanship of Preparatory Year and Supporting Studies, Imam Abdulrahman Bin Faisal University, Dammam 31441, Saudi Arabia
[2]Data Scientist, Applied Mathematics Research Engineer, National Engineering School of Tunis, Tunis El Manar University, Tunis 1068, Tunisia

Corresponding author: Minyar Sassi Hidri (mmsassi@iau.edu.sa)

ABSTRACT The sequential pattern mining field is expanding through numerous researches and has a large number of applications such as language processing, alarms management and event management on a broader scale. Its use began with processing items baskets to learn patterns and have a directed marketing strategy but it is generalized to telecommunication alarms management with several works. Our work is in line with this, as it tries to locate patterns and identify them to make predictive statements about certain patterns. It is axed around providing a way to break sequences into episodes and assigning them a value of confidence and support, more precisely in the discovery of frequent patterns of episodes within a time window. Experimental results have shown the effectiveness of our sequential pattern mining approach and its adaptability to alarm management and analytics.

INDEX TERMS Sequential pattern mining, alarm management, association rules, data mining, artificial intelligence.

I. INTRODUCTION

Data mining is defined as the science of extracting meaningful information from several flows of large data. It is about building empirical models on data-driven models, not with an underlying theory behind it. It aims to provide insight and useful visualization to its users, to reduce the complexity of huge databases. It is also referred to as Knowledge Discovery in Databases (KDD) [27]. There are several fields of application such as:

- Business Intelligence [40].
- Exploratory Data Analysis [1].
- Bioinformatics [22].
- Big Data [9], [36].
- Predictive Analytics [20].

Thus science groups a number of underlying methods and techniques, whose aims vary and spread all over a spectrum of applications: pattern recognition [4], statistical models [19], predictive analysis [20], machine learning [25], Information science [6], etc. It can be used in many different ways. Some of the tasks most commonly found are:

- Summarization & Reduction [35].
- Cluster Analysis [12], [18], [29].
- Regression Analysis [34].
- Classification and case-based reasoning [28].
- Association rule learning and sequential pattern mining [2], [24].

Industrial plants are an environment filled with data and it has become a critical issue to uncover knowledge in the large databases generated by the systems that are in play. The goal is that models built upon data mining will uncover relevant information that will help increase productivity, decrease potential risks, improve processes, pinpoint deviations in Key Performance Indicators (KPI) and aid the operators dealing with a huge overload of information. This is done by grouping key elements (clustering), providing a reduced dashboard through Human Machine Interfaces (HMI) with visualization, find recurring patterns to uncover cause-effect relationships and predict outputs or errors. And, in the new paradigm of industrialization and the new Industry 4.0 fashion, Data is at the essence of any industrious project. As well as collection and valorization which could be interpreted as feedback and monitoring, Data Analytics are starting to be critical: between getting correlations among interacting systems, and root-cause analysis, the techniques are numerous. Another bridge to cross in the future:

The associate editor coordinating the review of this manuscript and approving it for publication was Sunith Bandaru.

- Logs and records are raw *Data*. *Information* is the product of refining these data to obtain something relevant, *knowledge*. How to get there?
- How can we use past data to access knowledge about the future?
- Are frequent patterns in an alarm system (or even in a broader sense) indicator of relevant data? If so, how do we quantify that?

Since the number of alarms is growing and the monitoring process is getting harder as the operator has to acknowledge multiple alarms and deal with them, predictive models have become a necessity to assess the situation and set up contingency plans: alarms behavior is mapped and dealt with accordingly.

The application of data mining techniques to industrial engineering is a field that is getting more momentum, but that is currently underdeveloped. Data mining can, however, be strategically applied to industrial engineering processes such as scheduling, quality control, cost reduction, safety, and others.

There are various data mining techniques, most are data-specific. There are no *algorithms that apply on everything. e.g.* Neural network algorithms can be used on quantitative data, but adapting them to a qualitative data analysis is a quite rigorous and difficult task. Some of the techniques applied in data mining include traditional statistics. The predictive models are based on already logged data and generated using many algorithms that find frequent patterns that can pinpoint recurring problems. Also is considered that in dealing with said alarms, the operator creates his own set of patterns, that indicate how a problem A has been resolved following a solution pattern. This can improve reaction time but also be used in a manner to assist the operator if an issue presenting a degree of similarity to the one that has been resolved.

To do so, the system should be able to compute the similarity between patterns and align them correspondingly. Alignment and similarity are important, they allow to pinpoint patterns in dataflows with all data noise that may arise.

Our work is axed around alarm systems in industrial plants, the data being the alarms logs generated by the plant, more precisely in the discovery of frequent patterns of episodes within a time window.

Many approaches have been proposed to extract theses patterns to manage alarm systems [10], [11], [17].

The events of interest are the alarms triggering factors, their return to normal and whether they have been acknowledged by an operator or not. Some alarms floods can be defined briefly as the triggering of a high number of alarms in a small time window.

The remainder of this paper is organized as follows: in section II, we briefly present a state of the art on sequential pattern mining. Section III presents the preliminaries relates to our approach. Section IV is a bit more theoretical, as it presents our modeling approach related to sequential pattern mining. In section V, a performance analysis is brought forward to set the advantages of the implementation. Finally, we conclude in section VI our work and bring out some insights and potential future works.

II. RELATED WORK ON SEQUENTIAL PATTERN MINING

Several applications of data mining and machine learning emerge from analyzing data that comes from events sequences and alike data sets. Such data can come from alarms systems in different areas like telecommunications and industrial plants, natural language processing, users' actions on web sites and data flows on illnesses, etc.

In a more structured view, this data, which is streaming, can be viewed as a sequence of events ordered by a timestamp assigned to it. Events are viewed from a time and qualitative perspectives, both carry the data along with them.

To transform this data into meaningful information and knowledge, it has to be quantified and structured. This part of data mining has been gaining momentum since early 1990 and the interest is growing. From the pioneering work of Srikant and Agrawal [32] which has led to the Apriori family of algorithms and the definition of itemsets and association rules, to the later work of Mannila and Toivonen that has formalized episode mining and gave birth to the WINEPI (Sliding window approach for finding frequent episodes) / MINEPI (Minimal occurences approach for finding frequent episodes) family of algorithms.

Through 30 years of study and development, many techniques and approaches have been proposed for mining sequential patterns in a wide range of real-word applications [13], such as Web mining [3], classification [5], and mining motifs from biological sequences [8]. Some well-known serial algorithms for sequential pattern mining, such as AprioriAll [2], GSP algorithm (Generalized Sequential Pattern algorithm) [33], BIDE algorithm (BI-Directional Extension algorithm) [38], CloSpan (Software package of mining closed sequential patterns in a sequence database) [37], FreeSpan (**Fre**qu**e**nt Pattern-Projected **S**equential **pa**ttern Mining) [16], PrefixSpan (Prefix-projected Sequential pattern mining) [26], SPADE (Sequential PAttern Discovery using Equivalence classes) [39], etc., have been proposed. Since many parallel SPM (Sequential Pattern Mining) algorithms are extended by the traditional serial SPM approaches, it is quite necessary for us to have an understanding of the systematic characteristics of traditional serial sequential pattern mining approaches. Thus, the state-of-the-art of serial SPM are reviewed below. Based on the different mining mechanisms, some well-known serial SPM approaches are divided into several categories, including Apriori-based techniques, pattern growth techniques, algorithms with early-pruning, and constraint-based algorithms.

Each method of serial SPM algorithm has advantages and disadvantages. Besides, there are also many different definitions of categories for SPM algorithms. A more comprehensive discussion of these methods has been given in [15], [21], [24], and an up-to-date survey of the current status of SPM can be referred to [13].

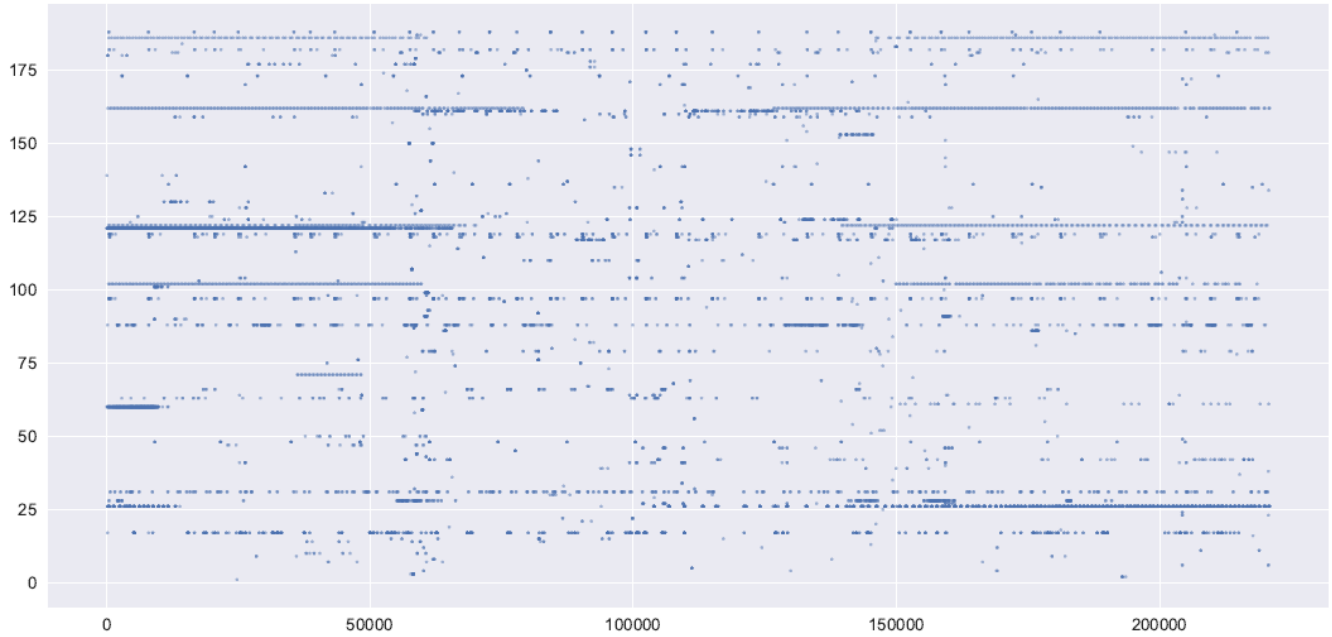


FIGURE 1. A scatter plot of an alarm log.

III. PRELIMINARIES

A. SCOPE

A pattern is defined as a repeated sign with regularity, as such, it has been linked to prediction as the regularity and frequency of repetition make it distinguishable.

It has been theorized that superior human intelligence comes certainly from its great pattern recognition features. Artificial Intelligence is considered to have surpassed it when it can recognize pattern better when a computer learned how to find patterns in chess, as would a chess grandmaster, it could beat him and surpasses him. Pattern Recognition is at the essence of machine learning and ranges from visual patterns to patterns in information. Visually, a pattern is a geometric figure that is repeated using geometric transformations, one of the greatest mathematical patterns are fractals. Our greatest feat is that we decomposed a seemingly infinite figure to a finite pattern that is repeated infinitely.

One of its features is our capacity to differentiate shapes, recognize people and extrapolate relations (knowing that two people are related or are from a certain country).

In data patterns, the approach is less obvious. Patterns in data sequences are not as much visual and the problems arise of how do we find those patterns. We can visualize data so that we can make those pattern geometric as shown in figure 1, but the problem arises of what if there are patterns that have no geometric visualization.

B. DEFINITIONS

To introduce the different methods and algorithms, we should firstly layout the various necessary definitions that formalize the work done in this field [32], [41].

Definition 3.1 (Event): Let E be the set of event types, called also an alphabet, and for the sake of simplicity,[1] the event types are a finite subset of $\mathbb{N}$ so that we have:

$$E = \{1, 2, \ldots, n\} \tag{1}$$

An event α_i is defined as the pair of an event type from the alphabet and occurrence time. The event can contain several other attributes but in this overview, we will limit ourselves to this pair:

$$\alpha_i = (e_i, t_i), \tag{2}$$

with $e_i \in E$ and t_i a positive integer denoting the time.

Definition 3.2 (Sequence): A sequence of data can be presented in the form of a database, a log file from the alarm plant system, etc. In this formalization, a sequence is defined as the succession of events sorted by their time of occurrence. The sequence itself is a triple (s, T_s, T_e) where T_s and T_e are respectively the start time and the end time of the sequence[2]:

$$s = \langle \alpha_1 \alpha_2 ... \alpha_n \rangle = \langle (e_1, t_1)(e_2, t_2)...(e_n, t_n) \rangle \tag{3}$$

with $\quad T_s \leq t_1 \leq t_2 \leq ... \leq t_n < T_e$

Example 3.1: On figure 2, we can represent the following sequence:

$$\begin{aligned}(\mathbf{s}, 30, 44) = \langle &(4, 31), (5, 32), (3, 33), (4, 33), (3, 35),\\ &(5, 35), (4, 37), (5, 38), (4, 39), (5, 39),\\ &(3, 40), (4, 40), (5, 40), (3, 41), (4, 42)\rangle\end{aligned}$$

[1] Without loss of generality, as any subset can be mapped to this one with any bijective function $f : \mathcal{A} \rightarrow E$

[2] An event with occurrence time equal to T_e is not counted in the sequence

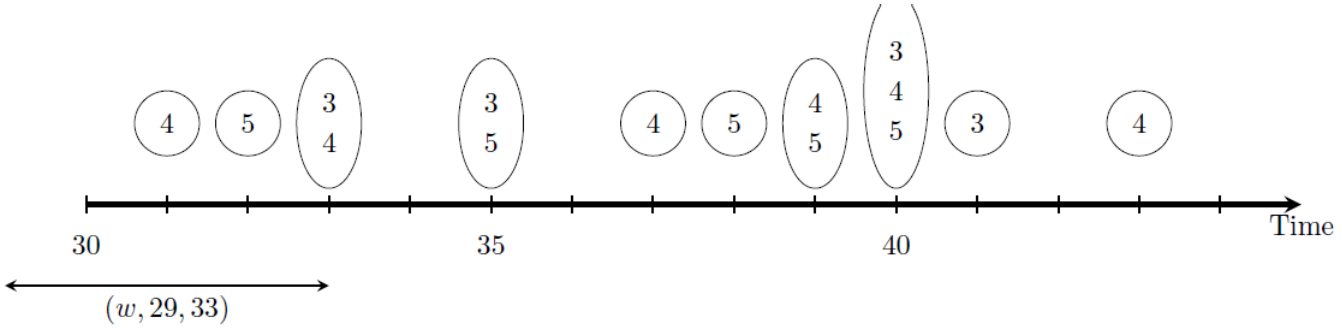


FIGURE 2. A representation of a sequence (in this example there are some events that occur at the same time).

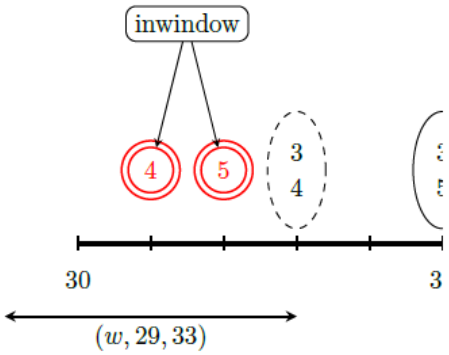


FIGURE 3. A detailed representation of a sequence.

We can see that events are happening at the same time, there are gaps in the sequence, (i.e. periods with no events happening). This is very convenient for our modeling purposes as the alarms can happen at the same times and there can be a duration in time where no alarm is triggered.

To be able to *count*, events occurrences and later *episodes* (which will be defined later on), we have to define when does occurrences count and how events are considered successive and/or close enough. For that purpose, the user should specify an *event window* and events in that window can be taken into consideration.

We define an event window as an event sequence:

$$\mathbf{w} = (w, t_s, t_e) \quad \text{where } t_s < T_e \text{ and } t_e > T_s \tag{4}$$

and by this definition, the window can extend outside of the sequence as shown in figure 2.

Example 3.2: The window $(w, 29, 33)$, as shown in figure 3, contains the events types: 4 and 5. It should be noted that the events (3, 33) and (4, 33) happened at the end of the window but as the sequence is defined: those shall not be counted. A representation of a sequence; in this example there are some events that occur at the same time

We call the width of a window its length in time:

$$(w, 29, 34) = (\langle (4, 31), (5, 32) \rangle, 29, 33) \tag{5}$$

$$width(\mathbf{w}) = t_e - t_s = 4 \tag{6}$$

$\mathcal{W}(\mathbf{s}, win)$ is the set of all windows of width win, it contains all the windows inside the sequence and those extending outside it, so it follow that:

$$|\mathcal{W}(\mathbf{s}, win)| = T_s - T_e + win - 1 \tag{7}$$

Definition 3.3 (Episodes): Previously, we defined sequences and events as the primary brick of this formalism, they represent the raw data, few modifications to the logs are done to get to this point, but to transform them into knowledge, we have to set up an essential part of the theory: Episode. We need to find the frequent pattern in the sequence that was provided to us, as such, events must organize into a meaningful structure that can be quantified as recurrent and/or happening. These are the episodes that are a class of event types, that are held up to a certain structure.

In a broader sense, an episode is an ordered (or partially ordered) collection of event types occurring together. Episodes can be described as directed acyclic graphs. They can be viewed as a relation mapping different event types, creating an antecedent/precedent relation between each event or a simultaneity relation.

Consider, for instance, episodes α, β and γ in the figure 4:

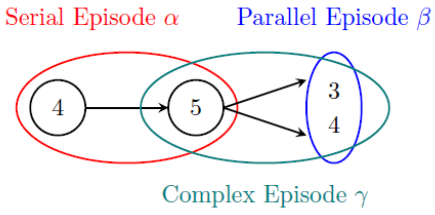


FIGURE 4. The different types of episodes.

Episode α is a serial episode: it occurs in a sequence only if there are events of types 4 and 5 that occur in this order in the sequence. In the sequence, there can be other events occurring between these two. The alarm sequence, for instance, is merged from several sources, and therefore it is useful that episodes are insensitive to intervening events.

Episode β is a parallel episode: no constraints on the relative order of 3 and 4 are given.

Episode γ is an example of the non-serial and non-parallel episode: it occurs in a sequence if there are occurrences of 3

and 4 and these follow an occurrence of 5; no constraints on the relative order of 3 and 4 are given. We mostly consider the discovery of serial and parallel episodes.

We now define episodes formally. An episode α is a triple $(V, \leq, g)$ where V is a set of nodes, $\leq$ is a partial order on V, and $g : V \to E$ is a mapping associating each node with an event type. The interpretation of an episode is that the events in $g(V)$ have to occur in the order described by $\leq$. The size of α, denoted $|\alpha|$, is $|V|$.

- Episode α is parallel if the partial order $\leq$ is trivial (i.e., $x \leq y \quad \forall x; y \in V$ such that $x \leq y$)
- Episode α is serial if the relation $\leq$ is a total order (i.e.,$x \leq y$ or $y \leq x \quad \forall x, y \in V$).
- Episode α is injective if the mapping g is an injection, i.e., no event type occurs twice in the episode.

To be able to specify whether an episode is recurrent, we define its frequency as the number of windows in which the episode occurs divided by the total number of windows $|\mathcal{W}(s, win)|$ on a sequence s.

Such as the frequency of an episode α in a sequence s with a window constraint *win* is:

$$fr(\alpha, s, win) = \frac{|\{w \in \mathcal{W}(s, win) \quad \text{such as } \alpha \text{ occurs in w }\}|}{|\mathcal{W}(s, win)|} \tag{8}$$

In this work, you will witness the influence of both works and how it is important in today's industrial revolution to have these tools that were formulated two decades ago provided that we adapt them to our needs in speed and efficiency.

We will set up a modeling approach to view the problems from a systems point-of-view and to get more perspective. The tools that are built in this work are alarm plant oriented but to preserve the generality of the work, we chose to separate the alarms part and the data mining part to get a more broader view of its applications.

IV. SEQUENTIAL PATTERN MINING: MODELING APPROACH

Now that the formalism is set and every aspect of the underlying theory is defined, we will move on developing a few algorithms that would encompass the goals of this work. These algorithms aim to find frequent patterns in sequences, and optimal alignment between those frequent *episodes*.

A. OPTIMAL ALIGNMENT PROCESSING

The Smith-Waterman algorithm was first proposed in [30]. Its objective is to find a pair of segments, one from each of two long sequences, such that there is no other pair of segments with greater similarity (homology)[3]

The Smith-Waterman algorithm is a local sequence alignment method. Before discussing the algorithm, we first introduce the concept of local alignment. Given a pair of symbolic segments, one from each of two symbolic sequences, we can equalize the length of the two segments by inserting gaps (symbol "-") in one or both of them (if the two segments have the same length, we can also choose to insert no gap). Then each symbol in one segment has a corresponding symbol in the other segment in the same position. This is called alignment, as shown in figure 5.

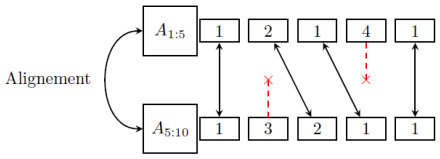


FIGURE 5. An alignement of two subsequences of a *A*, which could be identified as two alarms floods on the same database of alarms.

Since the two symbolic segments are two contiguous sub-sequences of the two symbolic sequences, respectively, the alignment on the pair of segments is called the local alignment of the two sequences.

An example of the similarity function used in this work is:

$$s(x, y) = \begin{cases} 1 & \text{if } x = y \\ -0.6 & \text{else} \end{cases} \tag{9}$$

If there is no alignment i.e. there is a gap, the penalty will be smaller $\delta = -0.4$ and for simplicity's sake it will be a uniform penalty (for successive gaps there are no additional penalty). Let $I(A_{p:q}, A_{k:l})$ be the similarity index of the alignment of sequences $A_{p:q}$ and $A_{k:l}$.

$$I(A_{p:q}, A_{k:l}) = \sum_{0 \leq i \leq \max(l-k, p-q)} s(A_{p+i}, A_{k+i}) \tag{10}$$

It represents the similarity of two sequences and the larger it is, the more similar the sequences are. It can be negative in cases which lead us to apply a constraint: if two sequences are completely dissimilar, their similarity score should be 0.

Let $S(A, B)$ be the similarity score of two sequences A and B, we have then:

$$S(A, B) = \max_{\substack{1 \leq i \leq p \leq M \\ 1 \leq j \leq q \leq N}} (I(A_{i:p}, B_{j:q}), 0) \tag{11}$$

The Smith-Waterman algorithm [30] makes use of a similarity matrix that is generated while iterating through every occurrence of both sequences.

This similarity matrix is then used to trace the optimal *route* which represents the best alignment strategy for the sequences: so after generating the matrix, all we have to do is backtrack from the best score and how it was computed. This matrix is generated using the following formula:

$$H_{p+1,q+1} = \max_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} (I(A_{i:p}, B_{j:q}), 0) \tag{12}$$

This can be used more easily in the recursive form which will help us use a dynamic programming approach to compute all terms of the matrix as in figure 6.

$$H_{p+1,q+1} = \max(H_{p,q} + s(a_p, b_q), H_{p,q+1} + \delta, H_{p+1,q} + \delta, 0) \tag{13}$$

[3]Smith and Waterman, 1981.

	Δ	1	3	2	1	1	2	3
Δ	0	0	0	0	0	0	0	0
1	0	1	0.6	0.2	1	1	0.6	0.2
2	0	0.6	0.4	1.6	1.2	0.8	2	1.6
1	0	1	0.6	1.2	2.6	2.2	1.8	1.4
4	0	0.6	0.4	0.8	2.2	2	1.6	1.2
1	0	1	0.6	0.4	1.8	3.2	2.8	2.4

$A_{1:5}$:	1	–	2	1	4	1		
Score	1	−0.4	1	1	−0.4	1	=	3.2
$A_{10:16}$:	1	3	2	1	–	1		

FIGURE 6. A test of a similarity of two sequences.

In the topic of alarm floods, we have to keep in mind that the sequence of data has a time factor that should be accounted for. Alignment is considered not on the sequence itself and its order but events that hold a type and a timestamp. This will change the score function as it should take into consideration the timestamp.

If we choose the alphabet of event types as[4]:

$$E = \{1, 2, ..., N\} \tag{14}$$

and the alarm sequence A as:

$$A = (A, t_1, t_M + 1) =< (e_1, t_1), ..., (e_M, t_M) >, \quad e_i \in E \quad i = 1..M \tag{15}$$

The idea behind this modification of the Smith-Waterman algorithm is to switch from a time-independent similarity function $s(x, y)$ to a time-dependant one $s((e_i, t_i), (e_j, t_j))$. This brings us to the need to define new concept: *time distance vector* and *time weighted vector*.

For an event (e_m, t_m) we define the time distance vector as follows:

$$\mathbf{d}_m = \left[d_m^1 \, d_m^2 \, ... \, d_m^N \right]$$

$$d_m^k = \begin{cases} \min\limits_{1 \le i \le M} \{|t_i - t_m| : e_i = k\} & \text{if the set is not empty} \\ \infty \end{cases}$$

These vectors are needed to see the proximity of events that hold the same type, thus the denomination *time distance vector*.

To homogenize the time distance vector and obtain weights needed for the similarity function, a mapping is used, a function $f(.)$ that has the following properties:

- $f : \mathbb{R}^+ \to [0, 1]$,
- f is monotonically decreasing,
- $f(0) = 1$ and $f(+\infty) = 0$.

[4]This is without loss of generality because a bijective function can map this alphabet to another with the same cardinality.

As an example here are a few weighting functions:

$$f(x) = \exp\left(-\frac{x^2}{\sigma^2}\right) \quad \text{(Gaussian distribution)}$$

$$f(x) = \exp(-\lambda x) \quad \text{(Exponential Distribution)}$$

$$f(x) = \frac{1}{1 + x^2} \quad \text{(Cauchy Distribution)}$$

$$f(x) = \delta(x) \quad \text{(Dirac Distribution)}$$

In this work, and for the sake of simplicity, we used the Gaussian distribution, as we can control its variance. Then, we defined the time weight vector as follows:

$$\mathbf{w}_m = [w_m^1 w_m^2 ... w_m^K] \tag{16}$$

$$= [f(w_m^1) f(w_m^2) .. f(w_m^K)] \tag{17}$$

We use the Gaussian distribution for the first sequence and the Dirac distribution for the second one. This makes the similarity function asymmetric: $s(A, B) \neq s(B, A)$. The similarity function is defined as follows:

$$s(A, B) = \max_{1 \le k \le K} [w_A^k \times w_B^k](1 - \mu) + \mu \tag{18}$$

B. SEQUENTIAL PATTERN MINING ALGORITHM WITH WINDOW CONSTRAINT

The work presented in [23] is a pioneering step forward in sequential pattern mining and opened the way to some improvements on the technique used to generate frequent episodes and differentiate between them. Many of the modern pattern recognition algorithms in sequences, (e.g. natural language recognition) have their roots from the WINEPI algorithm [23]. The name WINEPI is derived from the words: Window and Episode. This algorithm uses a window constraint to compute the support of an episode (see 3.3). This idea makes the generation of frequent and the rule discovery distinct tasks that could be done separately.

Simplified view of the Apriori Lemma.

Algorithm 1 recognizes candidate parallel episodes in a sequence, it proceeds as subsequently, when an event in a parallel episode α enters the current window, it is counted in $\alpha.eventCount$ which keep how much of this episode is in the window. Once $\alpha.eventCount = |\alpha|$, which indicates that episode α is entirely in the window, we save the starting time in $\alpha.inwindow$.

When the episode α leaves the window, as the $\alpha.eventCount$ decreases, it is saved in $\alpha.freqCount$ to mark how many windows the episode has remained entirely in, it will lead us to compute the support of the episode presented by the formula in 3.3.

In order to organize all candidates and make access efficient, it is useful to index them by a dictionary of how many events A, the key is the event type A and its number of occurrence in the episode, and the value is a list of episode having a number of event types A: this dictionary is called $contains[(A, a)]$, a being the number of occurrences.

Example 4.1: As an example in figure 7, suppose we have a window update and the new addition to the window is

Algorithm 1 Algorithm for recognizing parallel episode

Input: A collection $\mathcal{C}$ of parallel episode, an event sequence $s = (s, T_s, T_e)$, a window width *win*, and a frequency threshold *minFr*

Output: The frequent episode of $\mathcal{C}$ with respect to *win* and *minFr*

function FindParallelFrequent(s, $\mathcal{C}_l$, *win*, *minFr*)
▷ Initialization
  for all α in $\mathcal{C}$ **do**
    for all A in α **do**
      $A.count \leftarrow 0$
      for $i \leftarrow 1$ **do**$|\alpha|$
        $contains(A, i) \leftarrow \emptyset$
  for all $\alpha \in \mathcal{C}$ **do**
    for $\forall A \in \alpha$ **do**
      $a \leftarrow$ number of event of type A in α
      $contains(A, i) \leftarrow contains(A, i) \cup \{\alpha\}$
    $\alpha.eventCount \leftarrow 0$
    $\alpha.freqCount \leftarrow 0$
▷ Recognition
  for $start \leftarrow T_s - win + 1$ **do**T_e
    for all $(A, t) \subset s/t = start + win - 1$ **do**
      $A.count \leftarrow A.count + 1$
      for all $\alpha \in contains(A, A.count)$ **do**
        $\alpha.eventCount \leftarrow \alpha.eventCount + A.count$
        if $\alpha.eventCount = |\alpha|$ **then**
          $\alpha.inwindow \leftarrow start$
▷ Delete old event from window
    for all $(A, t) \subset s/t = start - 1$ **do**
      for all $\alpha \in contains(A, A.count)$ **do**
        if $\alpha.eventCount = |\alpha|$ **then**
          $\alpha.freqCount \leftarrow \alpha.freqCount - \alpha.inwindow + start$
        $\alpha.eventCount \leftarrow \alpha.eventCount - A.count$
  for all $\alpha \in \mathcal{C}$ **do**
    if $\frac{\alpha.freqCount}{T_e - Ts_s + win - 1} \geq minFr$ **then**
      return α

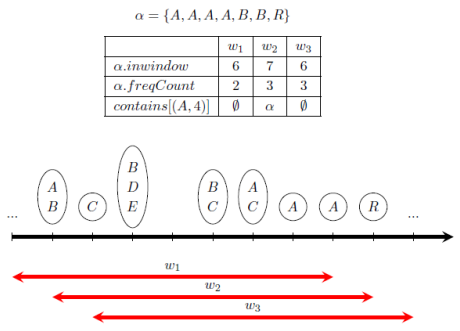


FIGURE 7. Parallel episode counting demonstration.

an event of type A, the window having 3 events of type A, the entry $contains[(A, 4)]$ is updated to signal to episodes expecting 4 A events, that this window contains them.

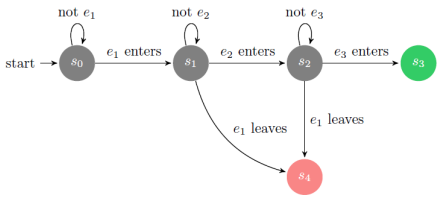


FIGURE 8. State automata recognizing a serial episode $\alpha =< e_1, e_2, e_3 >$.

The algorithm 2 is used for recognizing serial candidates episodes in an event sequence make use of state automata (like the one on figure 8). These automata that take candidates serial episodes as their sole input. For a serial episode α, we create an automaton, which we can instantiate many times simultaneously. These simultaneous instances can show the active states of the prefixes of α.

As the first event of a serial episode α enters the window, we create a new automaton for that episode; this automate is erased with the exit of the same first event from the time window.

The accepting state for an automaton is when all of its events have entered the window, with no other instance of this automaton in their accepting state. The starting time of the windows is saved in the variable $\alpha.inwindow$. This allows us to count the event once in a window since the frequency is the number of windows containing the episode at least once. In the situation where the accepting state automaton is exiting the window and when there is no accepting state automaton, we increase the variable $\alpha.freqCount$ with the number of windows in which the episode α has been in.

Having numerous automata with the same state is redundant, it provides the same information in this algorithm and would provide the same transitions. The optimal way is to keep the automaton that has to reach the most advanced state at the latest, as it will be removed last. At most, we should have $|\alpha|$ automata. To find the automaton that we should remove, we represent the automata with an array of size $|\alpha|$. In the variable $\alpha.initialized[i]$, we store the latest initialization time of an automaton in its i^{th} state. As α is a sorted array of events, this array is used to label the state transition.

To access and traverse the automata efficiently they are organized in the following way: in a list $waits(A)$ we store the automata that are waiting for an event $A \in E$; $waits(A)$ contains the episode and which event its automata is waiting for: (α, x) i.e. an automata for the x^{th} event of α. As event (A, t) enters the window, we consult the list $waits(A)$. If any automaton reaching the same state i as another, $\alpha.initialized[i]$ is overwritten to allow the latest state automaton to be saved.

We store transition at the shift of the window in a list, in the form (α, x, t): an episode α has reached its x^{th} event, with latest initialization time of the prefix of length x is t. We update old states of the automata immediately, new states are updated only after all the transitions have been pinpointed, only to not overwrite useful information. To easily remove automata instantiated at time t, we store them in a list $beginsat(t)$.

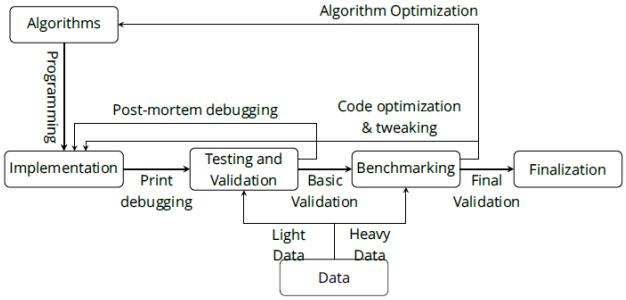


FIGURE 9. Our validation procedure.

	Δ	(1,0)	(3,1)	(2,4)	(1,4.2)	(4,7)	(3,9)	(2,12.5)
Δ	0	0	0	0	0	0	0	0
(4,0)	0	0	0	0	0	1	0.6	0.2
(1,3)	0	1	0.6	0.2	1	0.6	1.370	0.970
(3,5)	0	0.6	2	1.6	1.2	0.8	1.6	1.290
(1,7.5)	0	1	1.6	2.951	2.6	2.2	1.8	2.551
(2,8)	0	0.951	1.2	2.6	3.902	3.502	3.102	2.8
(3,13)	0	0.551	1.951	2.2	3.502	3.302	4.502	4.102
(1,20)	0	1	1.551	1.8	3.2	2.902	4.102	3.902

(a) Similarity table H

$A_{2:6}$: (1,3) (3,5) (1,7.5) (2,8) – (3,13)
Score: 1 + 1 + 0.951 + 0.951 + −0.4 + 1 = 4.502
$B_{1:6}$: (1,0) (3,1) (2,4) (1,4.2) (4,7) (3,9)

FIGURE 10. A test of a similarity of two sequences.

```
Number of rules found: 4

<Rule: 4 -> 4 4 / 0.3846153846153846>
<Rule: 4 -> 4 5 / 0.7692307692307692>
<Rule: 5 -> 4 5 / 0.9090909090909091>
<Rule: 5 -> 5 5 / 0.2727272727272727>
Time of execution: 0.004784 sec
```

FIGURE 11. A serial episode mining run on the example with window size *win* = 3, *minFr* = 0.2 and *minConf* = 0.1.

V. COMPUTATIONAL RESULTS

After presenting the underlying structure of the used algorithms and the theories they are based upon, it is crucial to test their implementation to validate the work done but also to have a showcase of performance indicators:

```
Number of rules found: 73

<Rule: 3 -> 3 4 / 0.8888888888888888>
<Rule: 3 -> 3 5 / 0.6666666666666666>
<Rule: 4 -> 3 4 / 0.6153846153846153>
<Rule: 4 -> 4 4 / 0.4615384615384615>
<Rule: 4 -> 4 5 / 0.6153846153846153>
...
<Rule: 5 4 5 5 -> 5 4 5 4 5 / 0.5>
<Rule: 5 5 4 5 -> 5 4 5 4 5 / 0.5>
Time of execution: 0.017568 sec
```

FIGURE 12. A serial episode mining run on the example with window size *win* = 3, *minFr* = 0.01 and *minConf* = 0.1.

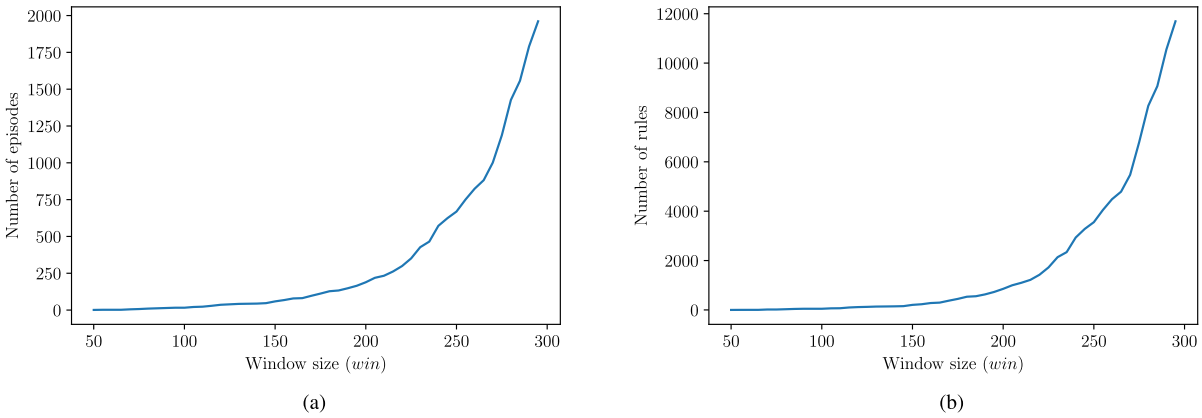
```
Number of rules found: 2515

<Rule: 288-PI-1272 -> 288-PI-1272 340-FI-434 / 0.7668530088786584>
<Rule: 288-PI-1272 -> 288-PI-1272 462-FI-1109 / 0.6941795462019072>
<Rule: 288-PI-1272 -> 288-PI-1272 462A-L52QC2 / 0.5771127918447878>
<Rule: 288-PI-1272 -> 288-PI-1272 462C-L52QC1 / 0.5656034199276554>
<Rule: 288-PI-1272 -> 288-PI-1272 462C-L52QC2 / 0.6885892798421571>
<Rule: 288-PI-1272 -> 288-PI-1272 IFC04-608INT / 0.6997698125616573>
<Rule: 340-FI-434 -> 288-PI-1272 340-FI-434 / 0.6959116681587586>
<Rule: 340-FI-434 -> 340-FI-434 340-FI-434 / 0.5601313040883318>
<Rule: 340-FI-434 -> 340-FI-434 462-FI-1109 / 0.7057594747836466>
<Rule: 340-FI-434 -> 340-FI-434 462A-L52QC2 / 0.5475977320202924>
<Rule: 340-FI-434 -> 340-FI-434 462C-L52QC1 / 0.5684870188003581>
<Rule: 340-FI-434 -> 340-FI-434 462C-L52QC2 / 0.6356311548791406>
....
<Rule: 462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109
    462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109
    462-FI-1109 -> 462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109
    462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109 462-FI-1109
    462-FI-1109 462-FI-1109 462-FI-1109 / 0.9687380861608845>
Time of execution: 8:9
```

FIGURE 13. First run of the algorithm on un-processed data: *win* = 300, *minFr* = 0.7.

- Stability;
- Time Consumption;
- Space Consumption;

These attributes are key indicators for algorithms that are bound to run on huge databases. Space and time consumption are as important as stability mainly because they affect it.

In this section, we aim to check the performance and the validity of such algorithms, as demonstrated in figure 9 and to extract the strengths of our procedure as well as the tweaks and optimizations applied.



(a) (b)

FIGURE 14. (a) Number of serial episodes, and (b)Number of rules generated as a function of the window size for serial episodes.

Algorithm 2 Algorithm for recognizing Serial Episode

Input: A collection $\mathcal{C}$ of serial episode, an event sequence $s = (s, T_s, T_e)$, a window width *win*, and a frequency threshold *minFr*

Output: The frequent episode of $\mathcal{C}$ with respect to *win* and *minFr*

function FindSerialFrequent($s, \mathcal{C}_l, win, minFr$)
▷ Initializing
  for all $\alpha \in \mathcal{C}$ **do**
    for $i \leftarrow 1$ **do**$|\alpha|$
      $\alpha.initialized[i] \leftarrow 0$
      $waits(\alpha[i]) \leftarrow \emptyset$
  for all $\alpha \in \mathcal{C}$ **do**
    $waits(\alpha[1]) \leftarrow waits(\alpha[1] \cup \{(\alpha, 1)\}$
    $\alpha.freqCount \leftarrow 0$
  for $t \leftarrow T_s - win$ **do**$T_s - 1$
    $beginstat(t) \leftarrow \emptyset$
▷ Recognizing Episodes
  for $start \leftarrow T_s - win$ **do**T_e
▷ Bring new events
    $beginstat(start + win - 1) \leftarrow \emptyset$
    $transitions \leftarrow \emptyset$
    for all $(A, t) \in s/t = start + win - 1$ **do**
      for all $(\alpha, j) \in waits(A)$ **do**
        if $j = |\alpha|$ and $\alpha.initialized[j] = 0$ **then**
          $\alpha.inwindow \leftarrow start$
        if $j = 1$ **then**
          $transitions \leftarrow transition \cup \{\alpha, 1, start + win - 1\}$
        else
          $transitions \leftarrow transition \cup \{\alpha, j, \alpha.initialized[j - 1]\}$
          $beginstat(\alpha.initialized[j - 1] \leftarrow beginstat(\alpha.initialized[j - 1] - \{(\alpha, j - 1)\}$
          $\alpha.initialized[j - 1] \leftarrow 0$
          $waits(A) \leftarrow waits(A) - \{(\alpha, j)\}$
    for all $(\alpha, j, t) \in transitions$ **do**
      $\alpha.initialized[j] \leftarrow t$
      $beginstat(t) \leftarrow beginstat(t) \cup \{(\alpha, j)\}$
      if $j < |\alpha|$ **then**
        $waits(\alpha[j+1]) \leftarrow waits(\alpha[j+1]) \cup \{(\alpha, j+1)\}$
      else
▷ Delete old events from window
    for all $(\alpha, l) \in beginsat(start - 1)$ **do**
      if $l = |\alpha|$ **then**
        $\alpha.freqCount \leftarrow \alpha.freqCount - \alpha.inwindow + start$
      else
        $waits(\alpha[l+1]) \leftarrow waits(\alpha[l+1]) - \{(\alpha, l+1)\}$
      $\alpha.initialized[l] \leftarrow 0$
  for all $\alpha \in \mathcal{C}$ **do**
    if $\frac{\alpha.freqCount}{T_e - T_s + win - 1} \geq minFr$ **then**
      return α

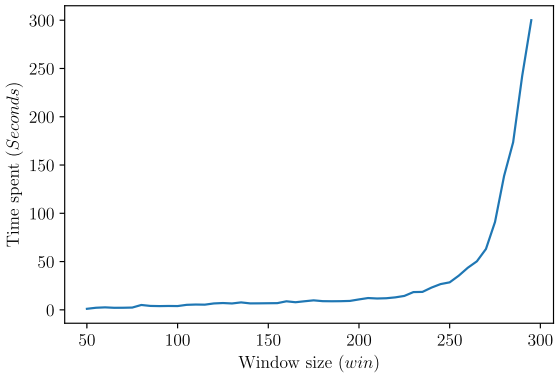


FIGURE 15. Run time as a function of the window size for serial episodes.

A. EXPERIMENTAL PROTOCOL

The test of the several algorithms is done on a laptop with an Intel i5 3230M CPU 2.60 GHz with 2 cores and 4 logical processors, a 4.0 GB of SODIMM DDR3 RAM @ 1600GHz.

The algorithms are programmed in Python 3.6 on an Anaconda 4.3.8 distribution and in C# using the Visual Studio 2015 distribution. Our validation procedure

The modified Smith-Waterman implementation aims to find alignment between two sequences. These sequences can be two subsets of the same data sequence, to compare two-alarm floods that follow the same pattern.

The modified Smith-Waterman algorithm uses a similarity score to pinpoint the best alignment of two sequences. The sequences are:

$$A = [(1, 0), (3, 1), (2, 4), (1, 4.2), (4, 7), (3, 9), (2, 12.5)]$$
$$B = [(4, 0), (1, 3), (3, 5), (1, 7.5), (2, 8), (3, 13), (1, 20)]$$

The algorithms presented in section IV-B have been implemented in Python 3 and tested on a few sample cases to be validated. The test cases range from simple text sequences to alarms databases.

To verify that our theoretical approach was on point, we started with a few test cases that will validate our implementation. Those test cases were text sequences and a succession of arbitrary events chosen randomly.

The next step is, of course, a performance showcase where the algorithms are put to a rigorous benchmark to see their computing ability. Those benchmarks are on an alarms database provided by Integration Objects.[5] It contains roughly 200000 events and a few thousand event types.

We, then, proceeded to plot the results in terms of numbers of episodes generated, rules generated and time consumed as a function of the several inputs of the algorithm.

B. RESULTS

The first part of our work to be tested is for optimal alignment. For the test case presented in the foundational paper [7], seen in figure 10 we found the same similarity score, which led us

[5]https://integrationobjects.com

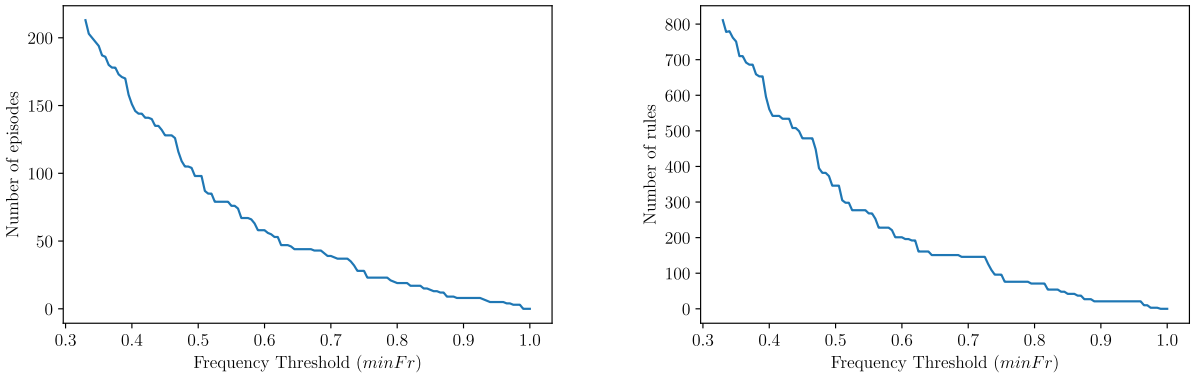


FIGURE 16. (a)Number of serial episodes, and (b)Number of rules generated as a function of the frequency threshold *minFr* for serial episodes.

to conclude on the correctness of our implementation. This was implemented in.NET C# under Visual Studio 2015.

The complexity of this algorithm is $\Theta(nm)$ and in that spirit, the lower bounds and upper bounds of complexity are the same. The performance of the algorithm is depending on the size of the sequences to be aligned. The matrix of correspondence is generated in its entirety every time the algorithm is running.

The whole focus of our implementation is to be useful in using data generated from the episode mining algorithm. It is a secondary objective for this work, but it can prove to be useful in ulterior research. Alignment can be used to find similar episodes and scoring sequences.

The second part of our work to be tested is for sequential pattern mining. This part of the algorithm is programmed in Python 3.6 under the same system. The rules are in the following form:

$$< Rule : \alpha \rightarrow \beta > \quad \text{with } \beta \prec \alpha \tag{19}$$

At first, we run the algorithm with data from the example in figure 11. The results are similar to those obtained from the manual computation which validates the correctness of our implementation.

The result was fairly fast and the number of rules is low since the frequency is considered high. A second run with a lower frequency would give us more rules as shown in figure 12.

The main objective of this algorithm is to run on a vast amount of data, we should benchmark its capabilities and how much resources it needs to run.

As a first run of the algorithm, we used a database provided by Integration Objects.[6] This has been done without pre-processing, to demonstrate the capabilities of the algorithms and to show that it can be used on alarms database.

We observe the need for pre-processing the data, as in figure 13, we have a chattering alarm that created a list of episodes that are quite frequent. Also, we can use this to pre-process the data in itself as the chattering alarms will create frequent patterns with only one event type.

[6]URL: https://integrationobjects.com.

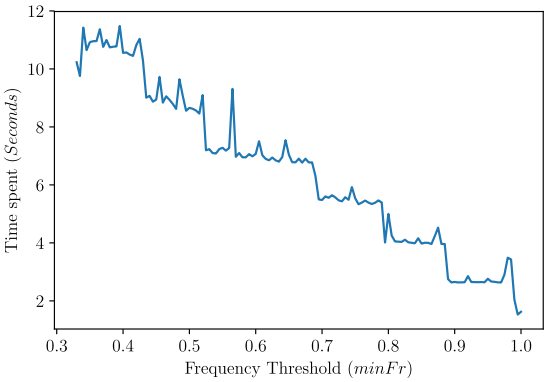


FIGURE 17. Run time as a function of the frequency threshold *minFr* for serial episodes.

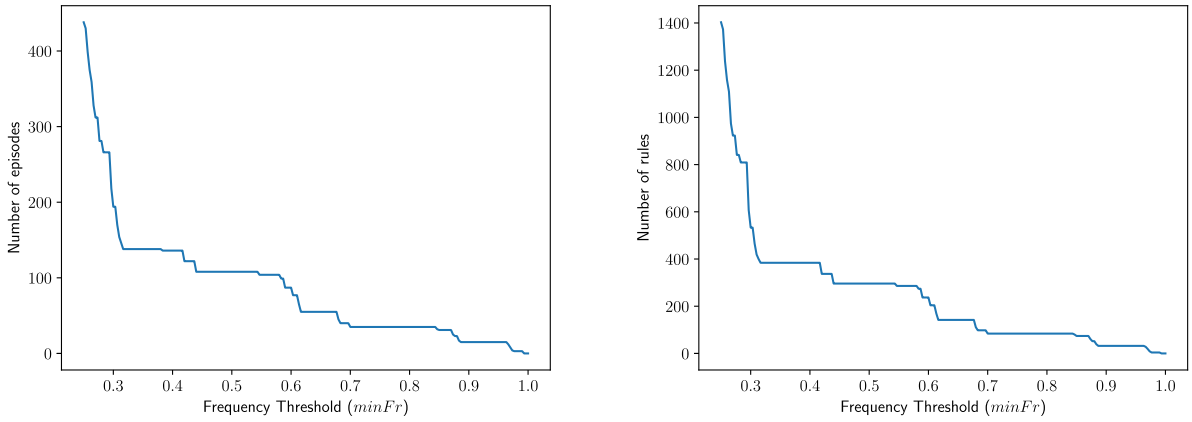
This run used up to 700 MB of memory and generated 2515 rules, with $T_s = 0$ and $T_e = 100000$ which represent roughly 28 hours of alarms data.

We will begin by testing the **serial episode generation algorithm**. As our implementation uses four algorithms and the algorithm that takes up most of the time is the candidate generation algorithm. We separate the testing for both implementations to benchmark performance.

In figure 14(a), we can observe that the number of episodes grows exponentially as a function of the window size. It is expected that episodes are an increasing function of the window size, since the bigger the windows the larger the spectrum of the search for episodes.

Like the previous statement, in figure 14(b), the number of episodes grows exponentially so the number of rules which is the number of relationships that episodes and their parents (super-episodes) share, i.e. confidence index.

In figure 15, we show the performance as the time spent to generate all episodes as a function of the window size. As stated in the theoretical analysis, it grows exponentially and window constraint should be chosen in a way that it is still generating enough meaningful data but on time.



(a) (b)

FIGURE 18. (a)Number of parallel episodes and (b)Number of rules generated as a function of the frequency threshold *minFr* for parallel episodes.

TABLE 1. Results of prediction-rule model for sequential rules, MineAlarmRulesForKind, and the proposed algorithm.

Algorithm	Prediction Count	Rule Occurrence Count	Accuracy	Recall
RuleGrowth	2,702	864	32.0%	69.8%
MineAlarmRulesForKind	366	266	72.7%	86.4%
Proposed algorithm %	**283**	**211**	**76.4%**	**91.1%**

In both figures 16(a) and 16(b), we have two monotonically decreasing plots of respectively, the number of episodes and the number of rules, with the frequency threshold as the variable.

The time plot in figure 17, as a function of the frequency threshold, is a step ladder like function. The time taken for generating episodes is the same for segments of time. Since the generation process is taken with the Apriori Principle, most of the runtime is for finding the 1-episodes and then generating the episodes from them.

The same process goes for **parallel episode generation**. We plot the figures with the frequency threshold and the time window as parameters and we see runtime, the number of episodes and rules generated.

In figures 18(a), 18(b),19,20(a), 20(b) we obtain similar results as the serial episode generation algorithms. There is a difference in the runtime as a function of the window constraint.

In figure 21, we see that we have a linear behavior which is a really interesting feat of the algorithm. This is due to the small size of the window. Its complexity has shown that it has to present quadratic like property but the implementation is more straightforward and process management in Python gave it a boost.

C. COMPARATIVE STUDY

After presenting the tests on the influence of system parameters, the proposed algorithm is compared with a traditional sequential data mining algorithm, RuleGrowth [14] and

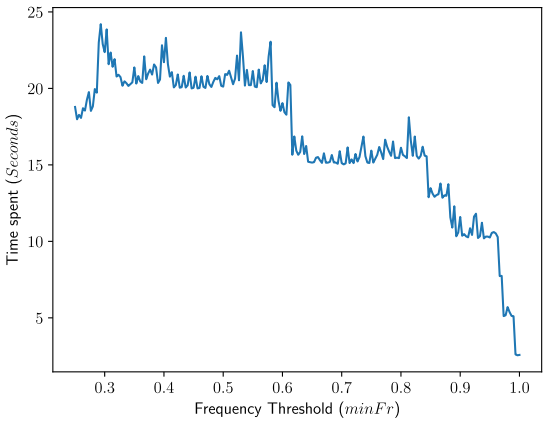


FIGURE 19. Run time generated as a function of the frequency threshold *minFr* for parallel episodes.

MineAlarmRulesForKind algorithm [31]. For this evaluation, the same prediction occurrence model with the alarm rules presented in [31] is implemented.

This model is based on the rule occurrences on the events. If the antecedent alarms of the sequential rule of a node happen in a transaction, then the prediction *the consequent alarm of this rule will also occur in this transaction* is made. If this consequent one also happens, the prediction is accurate. Then, the correctness of the prediction is calculated as accuracy. The alarm rules are mined from data set (alarm count: 369,345) which are dated from 10-08-2012 to 23-08-2012 based on the *Nokias* radio access network logs with a support value of 10 and confidence value of 0.9. The

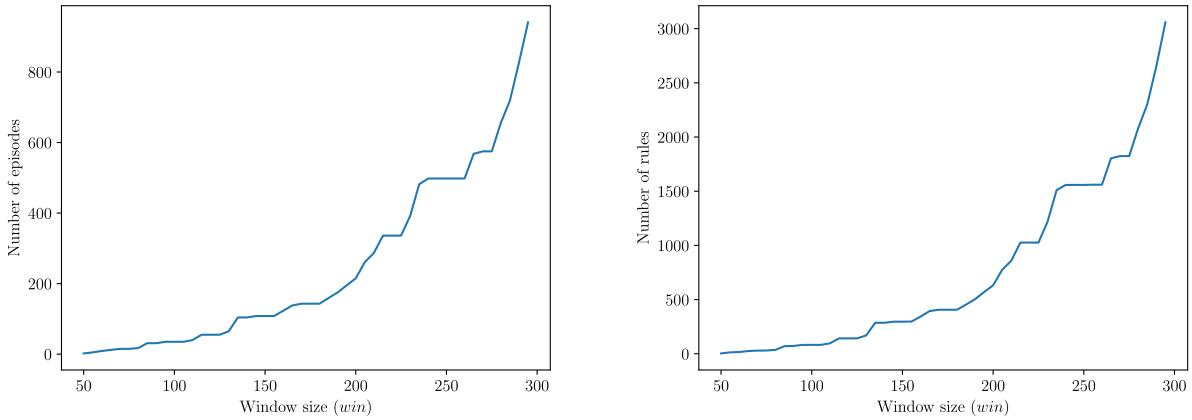


FIGURE 20. (a)Number of parallel episodes and (b)Number of rules generated as a function of the window size for parallel episodes.

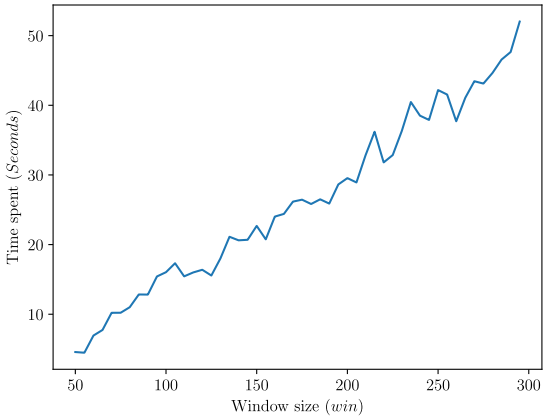


FIGURE 21. Run time as a function of the frequency threshold *minFr* for parallel episodes.

prediction accuracy is created from these logs (alarm count: 27,855) on 24-08-2012.

As shown in Table 1, the RuleGrowth algorithm makes 2,702 predictions and 864 rule occurrences take place based on these predictions in this experiment. Therefore, its accuracy is around 32.0%.

The MineAlarmRulesForKind algorithm makes 366 predictions and 266 rule occurrences take place based on these predictions in this experiment. Therefore, its accuracy is around 72.7%.

The proposed algorithm makes 283 predictions where 211 of them has occurred.

Therefore, the accuracy becomes 76.4%. In the experiments, the RuleGrowth and MineAlarmRulesForKind algorithms made many predictions while it utilizes a lot of resources and time.

These results show that the proposed method makes more precise predictions by using fewer resources and time.

VI. CONCLUSION

Thorough bibliographical research showed us that current literature considers sequential pattern recognition as an expanding field of research. The problem lies in its usage. The mining pattern from a single timed sequence of events is called: *episode* mining. This distinction helps us in refining our bibliographical research, finding and tweaking the existent algorithms.

In Data Mining, association rules is a machine learning tool to get information from sets of data but do not consider order (or time). Sequential pattern mining is finding the frequent/special pattern with no scoring method that has predictive capabilities. The algorithm present here was a reconciliation of both and the result obtained has probabilistic interpretation.

This work is axed around providing a way to break sequences into episodes and store those in a database with an assigned value of confidence and support. With this procedure, we can analyze data in a different way as episodes could have a certain meaning.

However, this approach still depends tightly on the choice of parameters as they will affect directly the extracted rules. In this context, further works can focus on making the choice of parameters more dynamic and dependent on the characteristics of data. Moreover, it is important to preprocess data due to noisy data, errors, inconsistencies, and lack of variable values. Different data preprocessing techniques like cleaning methods, data integration and transformation can be carried out before extracting rules to achieve successful analysis and prediction.

REFERENCES

[1] Y. Dodge, *Exploratory Data Analysis*. New York, NY, USA: Springer, 2008, pp. 192–194.
[2] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. 11th Int. Conf. Data Eng. (ICDE)*, Washington, DC, USA, 1995, pp. 3–14.
[3] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu, "Sequential pattern mining using a bitmap representation," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, New York, NY, USA, 2002, pp. 429–435.
[4] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics). Berlin, Germany: Springer-Verlag, 2006.
[5] S. Boggan and D. Pressel, "GPUs: An emerging platform for general-purpose computation," Comput. Inf. Sci., Army Res. Lab., Adelphi, MD, USA, Tech. Rep. ARL-SR-154, 2007.
[6] S. Y. Chen and X. Liu, "The contribution of data mining to information science," *J. Inf. Sci.*, vol. 30, no. 6, pp. 550–558, Dec. 2004.

[7] Y. Cheng, I. Izadi, and T. Chen, "Pattern matching of alarm flood sequences by a modified Smith–Waterman algorithm," *Chem. Eng. Res. Des.*, vol. 91, no. 6, pp. 1085–1094, Jun. 2013.

[8] D.-Y. Chiu, Y.-H. Wu, and A. Chen, "An efficient algorithm for mining frequent sequences by a new strategy without support counting," in *Proc. 20th Int. Conf. Data Eng.*, Washington, DC, USA, Sep. 2004, p. 375.

[9] J. Dean, *Big Data, Data Mining, and Machine Learning: Value Creation for Business Leaders and Practitioners*. Scotts Valley, CA, USA: CreateSpace Independent Publishing Platform, 2014.

[10] G. Dorgo and J. Abonyi, "Sequence mining based alarm suppression," *IEEE Access*, vol. 6, pp. 15365–15379, 2018.

[11] G. Dorgo, K. Varga, and J. Abonyi, "Hierarchical frequent sequence mining algorithm for the analysis of alarm cascades in chemical processes," *IEEE Access*, vol. 6, pp. 50197–50216, 2018.

[12] S. Brian Everitt, S. Landau, and M. Leese, *Cluster Analysis*, 4th ed. Hoboken, NJ, USA: Wiley, 2009.

[13] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of sequential pattern mining," *Data Sci. Pattern Recognit.*, vol. 1, no. 1, pp. 54–77, 2017.

[14] P. Fournier-Viger, R. Nkambou, and V. S.-M. Tseng, "RuleGrowth: Mining sequential rules common to several sequences by pattern-growth," in *Proc. ACM Symp. Appl. Comput. (SAC)*, TaiChung, Taiwan, Mar. 2011, pp. 956–961.

[15] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and S. P. Yu, "A survey of parallel sequential pattern mining," *ACM Trans. Knowl. Discov. Data*, vol. 13, no. 3, pp. 25:1–25:34, Jun. 2019.

[16] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "FreeSpan: Frequent pattern-projected sequential pattern mining," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, New York, NY, USA, 2000, pp. 355–359.

[17] R. Karoly and J. Abonyi, "Multi-temporal sequential pattern mining based improvement of alarm management systems," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Budapest, Hungary, Oct. 2016, pp. 3870–3875.

[18] R. S. King, *Cluster Analysis and Data Mining: An Introduction*. Herndon, VA, USA: Mercury Learning & Information, 2014.

[19] P. D. Kroese and J. C. C. Chan, *Statistical Modeling and Computation*. New York, NY, USA: Springer, 2013.

[20] D. T. Larose and C. D. Larose, *Data Mining and Predictive Analytics*, 2nd ed. Hoboken, NJ, USA: Wiley, 2015.

[21] N. R. Mabroukeh and C. I. Ezeife, "A taxonomy of sequential pattern mining algorithms," *ACM Comput. Surv.*, vol. 43, no. 1, pp. 1–41, Nov. 2010.

[22] A. M. Mabu, R. Prasad, R. Yadav, and S. S. Jauro, "A review of data mining methods in bioinformatics," in *Proc. Recent Adv. Eng., Technol. Comput. Sci. (RAETCS)*, Feb. 2018, pp. 1–6.

[23] H. Mannila, H. Toivonen, and A. Inkeri Verkamo, "Discovery of frequent episodes in sequences," Dept. Comput. Sci., Assoc. Adv. Artif. Intell., Univ. Helsinki, Helsinki, Finland, Tech. Rep., 1995.

[24] C. H. Mooney and J. F. Roddick, "Sequential pattern mining—Approaches and algorithms," *ACM Comput. Surv.*, vol. 45, no. 2, pp. 1–39, Feb. 2013.

[25] P. K. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.

[26] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "PrefixSpan: Mining sequential patterns by prefix-projected growth," in *Proc. 17th Int. Conf. Data Eng.*, Apr. 2001, pp. 215–224.

[27] G. Piatetsky-Shapiro, "Knowledge discovery in databases: 10 years after," *SIGKDD Explor. Newslett.*, vol. 1, no. 2, pp. 59–61, Jan. 2000.

[28] M. T. Rezvan, A. Z. Hamadani, and A. Shalbafzadeh, "Case-based reasoning for classification in the mixed data sets employing the compound distance methods," *Eng. Appl. Artif. Intell.*, vol. 26, no. 9, pp. 2001–2009, Oct. 2013.

[29] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin, "A review of clustering techniques and developments," *Neurocomputing*, vol. 267, pp. 664–681, Dec. 2017.

[30] T.-F. Smith and M.-S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, no. 1, pp. 195–197, Mar. 1981.

[31] S. E. Solmaz, B. Gedik, H. Ferhatosmanoğlu, S. Sözüer, E. Zeydan, and Ç. Ö. Etemoğlu, "ALACA: A platform for dynamic alarm collection and alert notification in network management systems," *Int. J. Netw. Manage.*, vol. 27, no. 4, p. e1980, Jul. 2017.

[32] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Advances in Database Technology*. London, U.K.: Springer-Verlag, 1996, pp. 1–17.

[33] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Proc. 5th Int. Conf. Extending Database Technol., Adv. Database Technol. (EDBT)*. London, U.K.: Springer-Verlag, 1996, pp. 3–17.

[34] S. Tuffry, *Data Mining and Statistics for Decision Making*, 1st ed. Hoboken, NJ, USA: Wiley, 2011.

[35] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2011.

[36] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[37] X. Yan, J. Han, and R. Afshar, "CloSpan: Mining: Closed sequential patterns in large datasets," in *Proc. SIAM Int. Conf. Data Mining*, May 2003, pp. 166–177.

[38] D. Yu, W. Wu, S. Zheng, and Z. Zhu, "BIDE-based parallel mining of frequent closed sequences with MapReduce," in *Algorithms and Architectures for Parallel Processing*, Y. Xiang, I. Stojmenovic, B. O. Apduhan, G. Wang, K. Nakano, and A. Zomaya, Eds. Berlin, Germany: Springer, 2012, pp. 177–186.

[39] M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Mach. Learn.*, vol. 42, nos. 1–2, pp. 31–60, Jan. 2001.

[40] M. J. Zaki and W. Meira, Jr., *Data Mining and Analysis: Fundamental Concepts and Algorithms*. New York, NY, USA: Cambridge Univ. Press, 2014.

[41] J. Zhong, W. Guo, and Z. Wang, "Study on network failure prediction based on alarm logs," in *Proc. 3rd MEC Int. Conf. Big Data Smart City*, Mar. 2016, pp. 1–7.

ADEL HIDRI was born in El Kef, Tunisia. He received the master's degree in automatic and signal processing, and the Ph.D. degree in electrical engineering from the National Engineering School of Tunis (ENIT), Tunis El Manar University, Tunisia, in 2004 and 2014, respectively. He is currently an Assistant Professor with the Imam Abdulrahman Bin Faisal University, Dammam, Saudi Arabia. His research is focused on multichannel speech separation and extraction, denoising speech, beamforming and microphone array, and machine learning.

AHMED SELMI was born in Tunis, Tunisia. He received the Engineering Diploma from the National Engineering School of Tunis, in 2017. He has worked as an IEEE Student Leader for a few years and was a member of different committees and the IEEE initiatives around the world, most notably internet initiative, ethical AI, and humanitarian connectivity.

MINYAR SASSI HIDRI was born in Nabeul, Tunisia. She received the degree in computer science engineering and the Ph.D. degree from the National Engineering School of Tunis (ENIT), Tunis El Manar University, Tunisia, in 2003 and 2007, respectively. She obtained the Habilitation to lead researches in computer sciences from Tunis El Manar University, Tunisia, in June 2014. She is currently an Associate Professor with ENIT, Tunisia, and an Assistant Professor with the Imam Abdulrahman Bin Faisal University, Dammam, Saudi Arabia, since September 2017. Her experience in teaching, in computer science and information systems is around 16 years. She had worked for five years as a Teaching Assistant at ENIT, and 10 years as an Assistant Professor. Besides her academic responsibilities, she had participated in several administrative tasks such that student's projects supervision and courses coordinator. Her research interests include combinatorial aspects in Big Data and their applications to different fields, including data mining, machine learning, deep learning, and text mining, with over 65 publications. She is also a member of the steering committee of many international conferences and a reviewer of impacted journals.