Event Forecasting with Pattern Markov Chains

Elias Alevizos
NCSR "Demokritos", Greece
alevizos.elias@iit.demokritos.gr

Alexander Artikis
[1]University of Piraeus, Greece
[2]NCSR "Demokritos", Greece
a.artikis@unipi.gr

George Paliouras
NCSR "Demokritos", Greece
paliourg@iit.demokritos.gr

ABSTRACT

We present a system for online probabilistic event forecasting. We assume that a user is interested in detecting and forecasting event patterns, given in the form of regular expressions. Our system can consume streams of events and forecast when the pattern is expected to be fully matched. As more events are consumed, the system revises its forecasts to reflect possible changes in the state of the pattern. The framework of Pattern Markov Chains is used in order to learn a probabilistic model for the pattern, with which forecasts with guaranteed precision may be produced, in the form of intervals within which a full match is expected. Experimental results from real-world datasets are shown and the quality of the produced forecasts is explored, using both precision scores and two other metrics: spread, which refers to the "focusing resolution" of a forecast (interval length), and distance, which captures how early a forecast is reported.

CCS CONCEPTS

• **Theory of computation** → *Formal languages and automata theory*; *Pattern matching*; *Random walks and Markov chains*; • **Information systems** → **Data streaming**;

KEYWORDS

Finite Automata, Regular Expressions

ACM Reference format:
Elias Alevizos, Alexander Artikis, and George Paliouras. 2017. Event Forecasting with Pattern Markov Chains. In *Proceedings of ACM International Conference on Distributed Event-Based Systems, Barcelona, Spain, June 19 – 23, 2017 (DEBS '17)*, 12 pages.
DOI: http://dx.doi.org/10.1145/3093742.3093920

1 INTRODUCTION

As analytics moves towards a model of proactive computing, the requirement for forecasting acquires more importance [7]. Systems with forecasting capabilities can play a significant role in assisting users to make smart decisions as soon as critical situations are detected. As an example, consider credit card fraud management. Automated fraud detection works with patterns consisting of long sequences of transactions with specific characteristics. Being able to forecast that part(s) of such sequences have high probability of leading to a full match (i.e., a fraud) can help an analyst focus on the involved cards and possibly take a proactive action even before the system detects the fraud.

The need for event forecasting as a means for proactive behavior has led to proposals about how forecasting could be conceptualized and integrated within a complex event processing system. However, such proposals still remain largely at a conceptual level, without providing concrete algorithms [6, 10]. On the other hand, there is a substantial body of work on the related field of time-series forecasting [20]. However, time-series analysis is usually applied on numerical data streams, where each element of the stream corresponds to a measurement of some variable of interest. Moreover, these measurements are often assumed to take place at time intervals of constant length. On the contrary, event processing systems need to be able to additionally deal with symbolic/categorical streams, where each element might be accompanied by arguments, either numerical or symbolic, arriving at unspecified timepoints.

We present an implementation of a system for event forecasting. We assume that event patterns are defined through regular expressions. As a first step, these patterns are converted to finite automata for the purpose of pattern matching. Subsequently, these automata are converted into Markov chains, which allow for the construction of a probabilistic model for the initial pattern. The final goal is to be able to forecast, as events arrive at the system, when the pattern will be fully matched. This is the first time that Pattern Markov Chains are used for online event forecasting. We show that our system can indeed forecast the completion of patterns in real-world datasets and that, under certain assumptions, it can do so with guaranteed precision. Moreover, we explore the quality of the produced forecasts, using three different metrics: precision score, spread, which refers to how focused a forecast is, and distance, which captures how early a forecast is reported.

The structure of the paper is as follows: Section 2 presents related work. In Section 3, the necessary mathematical terminology and framework are described. Section 4 elaborates on the implementation details of the system, while Section 5 presents experimental results on real-world datasets. Finally, in Section 6 we conclude with a summary and a discussion on future directions of research.

2 RELATED WORK

Timewewaver is a genetic algorithm that tries to learn from sequences of events a set of predictive patterns [26]. Its focus is on learning patterns that can forecast, as early as possible, rare events, such as equipment failures. In [4], the forecasting problem is formulated as a classification problem and the goal is again to construct predictive patterns for rare events. The proposed algorithm constructs a matrix of features, finds a reduced set of features through Singular Value Decomposition and then trains a set of Support Vector Machines, one for each target event.

DEBS '17, Barcelona, Spain

DOI: http://dx.doi.org/10.1145/3093742.3093920

Table 1: Methods for event forecasting (in order of publication date).

Paper	Prob.	Learning	Language	Relations	Focus
[11]	no	Syntactical pattern recognition	Chronicles (*before*, *equal*, *after* + numerical time constraints)	Only on timestamps	Expected events following a partial match
[26]	no	Genetic	Sequences (+ OR operator and wildcard events)	Discrete arguments. Equality and "don't care" operators on single events (no relations between different events).	Rare events
[4]	no	Singular Value Decomposition + Support Vector Machines	Feature matrix	Discrete arguments. Equality on single events (no relations between different events).	Rare events
[25]	no	Association (predictive) rule mining	Sets of events (not necessarily in sequential order)	no	Rare events
[16]	yes	Frequent episode discovery + Expectation Minimization	Sequences	no	Immediately next event
[27]	no	Decision trees	Sequences	no	Class labels of sequences
[2]	no	no	Directed Acyclic Graphs	(In)equality on single events (no relations between different events)	Minimal occurrences
[21]	yes	Learns probabilistic model. No pattern mining.	Sequences (+ *conjunction* and *disjunction*)	yes	Completion time of pattern
[12]	yes	Decision trees + Conditional Intensity Models	Sequences	no	Event sequences
[8]	no	Frequent episode discovery (starts from the consequent)	Sequences	no	Minimal antecedent, distant consequent
[29]	no	Sequential pattern mining	Sequences	no	Online update of patterns
Our approach	yes	Learns probabilistic model. No pattern mining.	Regular expressions	no	Completion time of pattern

A significant number of forecasting methods comes from the field of temporal pattern mining, where patterns are usually defined either as association rules [1] or as frequent episodes [19]. For example, in [25], a framework similar to that of association rule mining is used in order to identify sets of event types that frequently precede a rare, target event within a temporal window. In [16], a probabilistic model is presented. The goal is to calculate the probability of the immediately next event in the stream through a combination of standard frequent episode discovery algorithms, Hidden Markov Models and mixture models. Episode rules constitute the framework of [8] as well, where the goal is to mine predictive rules whose antecedent is minimal (in number of events) and temporally distant from the consequent. The algorithms presented in [29] focus on batch, online mining of sequential patterns, without maintaining exact frequency counts. At any time, the learned patterns (up to that time) can be used to test whether a prefix matches the last events seen in the stream and therefore make a forecast.

In [27], a variant of decision trees is used in order to learn sequence prefixes that are as short as possible and that can forecast the class label of the whole sequence. The method proposed in [2] starts with a given episode rule (as a Directed Acyclic Graph) and the goal is to build appropriate data structures that can efficiently detect the minimal occurrences of the antecedent of a rule defining a complex event, i.e., those "clusters" of antecedent events that are closer together in time. In [12], Piecewise-Constant Conditional Intensity Models and decision trees are employed in order to learn a very fine-grained model of the temporal dependencies among events in sequences. The learned models can then be used to calculate whether a sequence of target events will occur in a given order and in given time intervals. One of the earliest methods for forecasting is the Chronicle Recognition System, proposed in [5, 11], where events may be associated with both temporal operators and with numerical constraints on their timestamps. The system uses partial matches in order to report when the remaining events are expected for the pattern to complete. However, such forecasts are not based on a confidence or probability metric.

The work most closely related to ours is the one presented in [21], where Markov chains are also used in order to estimate when a pattern is expected to be fully matched. This work can also handle some relational constraints. On the other hand, the framework of Pattern Markov Chains that we use offers three main advantages: first, it can handle arbitrary regular expressions (and not only sequential patterns); second, it can handle streams generated by higher-order processes; third, it automatically calculates the expected time interval of pattern completion.

Table 1 summarizes the methods presented in this section. Its second column (*Prob.*) indicates whether a method employs a probabilistic framework. The third column (*Learning*) shows whether (and how) a method can automatically extract such patterns by reading (part of) the input event stream. The next two columns (*Language* and *Relations*) refer to the expressivity of the (learned or given) patterns. The entries in the *Language* column show how the different events in a pattern may be temporally related. The *Relations* column mentions if (and how) arguments of the involved events may be constrained and related. Finally, the goal of the last column is to show on what kind of forecasts each method focuses.

The last row shows how our approach compares to other methods. The advantage of our approach is that it moves beyond simple sequential patterns and combines the expressive power of regular

expressions with a rigorous probabilistic framework. The focus in this paper is on estimating when a full match of a given pattern will be detected. Note, however, that this does not exclude the possibility of incorporating (some of) the extra functionality of other methods. For example, frequent pattern mining could be a possible future extension by using the theory of Markov chains in order to estimate the expected number of occurrences of a pattern. One of the most important challenges for all methods (including ours) is *relationality*, i.e., the ability to handle patterns in which the arguments of *different* events in the pattern are related through some constraints.

3 THEORETICAL BACKGROUND

The problem we address could be stated as follows. Given a stream of events S and a pattern R, the goal is two-fold. First: find the full matches of R in S. Second: as the stream is consumed by the engine, forecast the full matches before they are detected by the recognition engine. For the recognition task, we use finite automata, whereas for forecasting, we convert these automata into Markov chains.

We make the following assumptions:

- Patterns are defined in the form of regular expressions.
- The *selection strategy* is either that of *contiguity* (i.e., events in a match must be contiguous, without irrelevant events intervening) or *partition-contiguity* (i.e., same as *contiguity* but stream may be partitioned by a specific event attribute). The *counting policy* is that of *non-overlap* (i.e., after a full match, the automaton returns to its start state). See [28] for the various selection strategies and [17] for the counting policies.
- The stream is generated by a m-order Markov process.
- The stream is stationary, i.e., its statistical properties remain the same. Hence the constructed Markov chain is *homogeneous* and its transition matrix remains the same at all time-points.
- A forecast reports for how many "points" we will have to wait until a full match. By the term "point", we refer to number of transitions of the Markov chain (or equivalently to number of future events) and not to time-points. Points are indeed time-points only in cases where a new event arrives at each time-point.

The theoretical tools presented in this section are based mostly on the work described in [9, 22, 23] and are grounded in the field of string pattern matching. For a review of this field, the reader may consult [17]. Comprehensive treatments of this subject may be found in [3, 13, 18].

3.1 Event Recognition

In this section, we briefly review some of the necessary terminology [14]. Regular expressions define the so-called regular languages. Within the context of the theory of regular languages, an *alphabet* $\Sigma = \{e_1, ..., e_r\}$ is a finite, non-empty set of symbols. The alphabet essentially refers to the set of the different event types that may appear in the stream. A string over Σ is a finite sequence of symbols from the alphabet. A language L over Σ is a set of strings over Σ. One common way to denote languages over Σ is through the use of regular expressions. If R denotes a regular expression, then $L(R)$ denotes the language defined by R. There are three operators that are used in regular expressions: *union*, which is binary and is denoted by the symbol +, *concatenation*, again binary, denoted by $\cdot$, and *star closure*, which is unary, denoted by *. Regular expressions are inductively defined as follows:

- The *union* of two languages L and M, $L \cup M$ is the set of strings that belong either to L or M. If R_1 and R_2 are regular expressions, then $R_1 + R_2$ is also a regular expression and $L(R_1 + R_2) = L(R_1) \cup L(R_2)$. Union corresponds to the *OR* operator in event recognition.
- The *concatenation* of two languages L and M, $L \cdot M$ is the set of strings formed by concatenating strings from L with strings from M, i.e., $L \cdot M = \{s_1 \cdot s_2, s_1 \in L, s_2 \in M\}$. If R_1 and R_2 are regular expressions, then $R_1 \cdot R_2$ is also a regular expression and $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2)$. Concatenation corresponds to the *sequence* operator in event recognition.
- The *star closure* of a language L is $L^* = \bigcup_{i \geq 0} L^i$, where L^i is concatenation of L with itself i times. If R is a regular expression, then R^* is also a regular expression and $L(R^*) = (L(R))^*$. Star closure corresponds to the *iteration* operator in event recognition.

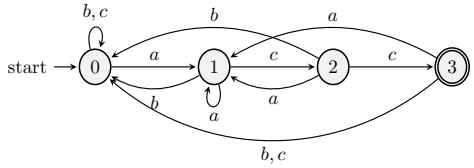
Finally, the inductive basis for a regular expression is that it may also be the empty string or a symbol from Σ.

Regular expressions may be encoded by deterministic and non-deterministic finite automata (DFA and NFA respectively). A DFA is 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ a finite set of symbols, $\delta : Q \times \Sigma \rightarrow Q$ a transition function from a state reading a single symbol to another state, $q_0 \in Q$ a start state and $F \subset Q$ a set of final states. A string $s = e_1 e_2 ... e_d \in \Sigma^*$ is accepted by the DFA if $\delta(q_0, s) \in F$, where the transition function for a string is defined as $\delta(q, e_1 e_2 ... e_d) = \delta(\delta(q, e_1 e_2 ... e_{d-1}), e_d)$. The definition for a NFA is similar with the modification that the transition function is now $\delta : Q \times \Sigma \rightarrow S_Q$, where S_Q is the power set of Q.

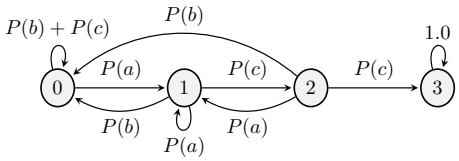
There exist well-known algorithms for converting a regular expression R to an equivalent NFA, NFA_R, and subsequently to an equivalent DFA, DFA_R [14]. For event recognition, a slight modification is required so that the DFA can detect all the full matches in the stream. The regular expression and DFA that should be used are $\Sigma^* \cdot R$ and $DFA_{\Sigma^* \cdot R}$ respectively so that the DFA may recognize all the strings ending with R [3, 13, 22]. Figure 1a shows an example of a DFA, constructed for $R = a \cdot c \cdot c$ (one event of type a followed by two events of type c) and $\Sigma = \{a, b, c\}$ (three event types may be encountered, a,b and c).

3.2 Event Forecasting

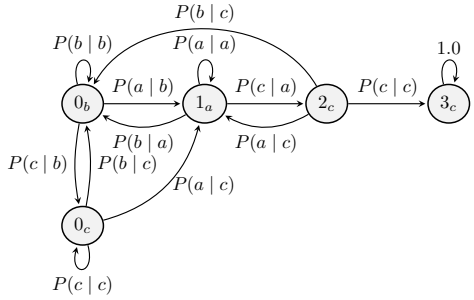
We use Pattern Markov Chains, i.e., convert $DFA_{\Sigma^* \cdot R}$ to an "appropriate" Markov chain. The Markov chain should be "appropriate" in the sense that it could be used in order to make probabilistic inferences about the run-time behavior of $DFA_{\Sigma^* \cdot R}$. In the case where the stream consumed by $DFA_{\Sigma^* \cdot R}$ is assumed to be composed of a sequence of independent, identically distributed (i.i.d.) events from Σ, then constructing the corresponding Markov chain is straightforward. As shown in [23], if $X = X_1, X_2, ..., X_i, ...$ is the i.i.d. sequence of input events, then the sequence $Y = Y_0, Y_1, ..., Y_i, ...$, where $Y_0 = q_0$ and $Y_i = \delta(Y_{i-1}, X_i)$ (i.e., the sequence of the states



(a) $DFA_{\Sigma^* \cdot R}$



(b) PMC_R^0



(c) PMC_R^1

Figure 1: DFA and PMCs for $R = a \cdot c \cdot c$, $\Sigma = \{a, b, c\}$ and for $m = 0$ and $m = 1$.

that $DFA_{\Sigma^* \cdot R}$ visits) is a 1-order Markov chain. Such a Markov chain, associated with a pattern R, is called a Pattern Markov Chain (PMC). Moreover, the transition probabilities between two states are simply given by the occurrence probabilities of the event types. If $p, q \in Q$ and $\mathbf{\Pi}$ is the $|Q| \times |Q|$ matrix holding these probabilities, then $\mathbf{\Pi}(p,q)$=$P(X_i$=$e)$, if $\delta(p,e) = q$ (otherwise, it is 0). This means that we can directly map the states of $DFA_{\Sigma^* \cdot R}$ to the states of a PMC and for each edge of $DFA_{\Sigma^* \cdot R}$ labeled with $e \in \Sigma$, we can insert a transition in the PMC with probability $P(e)$ (assuming here stationarity, i.e., $P(X_i = e) = P(X_j = e), \forall i, j$). As an example, Figure 1b shows the PMC constructed for $R = a \cdot c \cdot c$, based on the $DFA_{\Sigma^* \cdot R}$ of Figure 1a (the reason why state 3 has only a self-loop with probability 1.0 will be explained later).

For the more general case where the process generating the stream is of a higher order $m \geq 1$, the states of the PMC should be able to remember the past m symbols so that the correct conditional probabilities may be assigned to its transitions. However, the states of $DFA_{\Sigma^* \cdot R}$ do not hold this information. As shown in [22, 23] we can overcome this problem by iteratively duplicating those states of $DFA_{\Sigma^* \cdot R}$ for which we cannot unambiguously determine the last m symbols that can lead to them and then convert it to a PMC. From now on, we will use the notation PMC_R^m to refer to the Pattern Markov Chain of a pattern R and order m. Please, note that, from a mathematical point of view, the resulting Markov chain is always of order 1, regardless of the value of m [23].

As an example, see Figure 1c which shows the resulting PMC_R^1 for $R = a \cdot c \cdot c$. Note that the DFA for the same pattern with m=0, shown in Figure 1a, has a state which is ambiguous. When in state 0, the last symbol read may be either b or c. For all the other states, we know the symbol that led to them. Therefore, state 0 must be duplicated and state 0_c is added. Now, when in state 0_b, we know that the last symbol was b, whereas in state 0_c, it was c.

Once we have PMC_R^m for a user-defined pattern R, we may use the whole arsenal of Markov chain theory to make certain probabilistic inferences about R. For the task of forecasting, a useful distribution that can be calculated is the so-called waiting-time distribution. The waiting-time for a pattern R when its $DFA_{\Sigma^* \cdot R}$ is in state q is a random variable, denoted by $W_R(q)$. It is defined as the number of transitions until its first full match, i.e., until the DFA visits for the first time one of its final states.

$$W_R(q) = inf\{n : Y_0, Y_1, ..., Y_n, Y_0 = q, q \in Q \backslash F, Y_n \in F\}$$

The DFA is in a non-final state q and we are interested in the smallest time index $n > 0$ (i.e., first time) at which it will visit a final state. Informally, what we want to achieve through $W_R(q)$ is the following: each time the DFA is in some non-final state q (regardless of whether it is the start state), we want to estimate how many transitions we will have to wait until it reaches one of its final states, i.e., until a full match is detected. This number of future transitions may be given to the user as a forecast and it is constantly revised as more symbols are consumed and the DFA moves to other states. As a random variable, $W_R(q)$ follows a probability distribution and our aim is to compute this distribution for every non-final state q.

We can compute the distribution of $W_R(q)$ through the following technique. First, we convert each state of PMC_R^m that corresponds to a final state f of $DFA_{\Sigma^* \cdot R}$ ($f \in F$, with $|F|$=k) into an absorbing state, i.e., a "sink" state with probability of staying in the same state equal to 1.0 (state 3 in Figure 1). We can then re-organize the transition matrix as follows:

$$\mathbf{\Pi} = \begin{pmatrix} \boldsymbol{N} & \boldsymbol{C} \\ \mathbf{0} & \boldsymbol{I} \end{pmatrix} \tag{1}$$

where $\boldsymbol{I}$ is the identity matrix of size $k \times k$, corresponding to the absorbing states. If PMC_R^m has a total of l states (k of which are final), then $\boldsymbol{N}$ would be of size $(l-k) \times (l-k)$ and would correspond to the non-final states, holding the probabilities for all the possible transitions between (and only between) the non-final states. Finally, $\boldsymbol{C}$ is a $(l-k) \times k$ matrix holding the transition probabilities from non-final to final states and $\mathbf{0}$ is a zero matrix of size $k \times (l-k)$. For example, for the PMC of Figure 1b, the transition matrix would be the following:

$$\mathbf{\Pi} = \begin{matrix} \{0 \\ 1 \\ 2 \\ 3 \end{matrix} \begin{pmatrix} P(b)+P(c) & P(a) & 0 & 0 \\ P(b) & P(a) & P(c) & 0 \\ P(b) & P(a) & 0 & P(c) \\ 0 & 0 & 0 & 1.0 \end{pmatrix}$$

where, to the left of the matrix, for each of its rows, we show the corresponding states (in curly brackets). In this case, l=4, k=1 and $\boldsymbol{N}$ is of size 3×3. Through this re-arrangement, we can use the following theorem [9]:

Theorem 3.1. *Given a transition probability matrix* $\mathbf{\Pi}$ *of a homogeneous Markov chain* Y_t *in the form of Eq.* (1)*, the probability for the time index n when the system first enters the set of absorbing states can be obtained from*

$$P(Y_n \in A, Y_{n-1} \notin A, \ldots, Y_1 \notin A \mid \boldsymbol{\xi_{init}}) = \boldsymbol{\xi}^T \boldsymbol{N}^{n-1}(\boldsymbol{I} - \boldsymbol{N})\mathbf{1} \quad (2)$$

A denotes the set of absorbing states. $\mathbf{1}$ is simply a $(l-k)\times 1$ vector with all its elements equal to 1.0. $\boldsymbol{\xi_{init}}$ is the initial distribution on the states, i.e., it is a vector whose element i holds the probability that the PMC is in state i at the start. $\boldsymbol{\xi}$ consists of the $l-k$ elements of $\boldsymbol{\xi_{init}}$ corresponding to non-absorbing states.

In the theory of Markov chains, the current state of the chain is not always known and must be encoded in such a vector. For example, for the PMC of Figure 1b, we could have $\boldsymbol{\xi}^T_{\boldsymbol{init}} = (0.2\ 0.3\ 0.4\ 0.1)$, meaning that we are in state 0 with probability 20%, in state 1 with probability 30%, etc. However, in our case, at each point, the current state of $DFA_{\Sigma^*\cdot R}$ (and therefore of PMC^m_R) is known and therefore this vector would have 1.0 as the value for the element corresponding to the current state (and 0 elsewhere). $\boldsymbol{\xi}$ changes dynamically as the DFA/PMC moves among its various states and every state has its own $\boldsymbol{\xi}$, denoted by $\boldsymbol{\xi_q}$:

$$\boldsymbol{\xi_q}(i) = \begin{cases} 1.0 & \text{if row } i \text{ of } N \text{ corresponds to state } q \\ 0 & \text{otherwise} \end{cases}$$

A slight variation of Equation 2 then gives the probability of the waiting-time variable:

$$P(W_R(q) = n) = \boldsymbol{\xi_q}^T \boldsymbol{N}^{n-1}(\boldsymbol{I} - \boldsymbol{N})\mathbf{1}$$

4 IMPLEMENTATION

We implemented a forecasting system, Wayeb, based on Pattern Markov Chains. Algorithm 1 presents in pseudo-code the steps taken for recognition and forecasting. Wayeb reads a given pattern R in the form of a regular expression, transforms this expression into a NFA and subsequently, through standard determinization algorithms, the NFA is transformed into a *m-unambiguous* DFA (line 1 in Algorithm 1). For the task of event recognition, only this DFA is involved. At the arrival of each new event (line 7), the engine consults the transition function of the DFA and updates the current state of the DFA (line 8). Note that this function is simply a look-up-table, providing the next state, given the current state and the type of the new event. Hence, only a memory operation is required.

4.1 Learning the matrix of the PMC

To perform event forecasting, we need to create PMC^m_R and estimate its transition matrix. This is achieved by using the maximum-likelihood estimators for the transition probabilities of the matrix [18]. Let $\mathbf{\Pi}$ denote the transition matrix of a 1-order Markov chain, $\pi_{i,j}$ the transition probability from state i to state j and $n_{i,j}$ the number of transitions from state i to state j. Then, the maximum likelihood estimator for $\pi_{i,j}$ is given by:

$$\hat{\pi}_{i,j} = \frac{n_{i,j}}{\sum_{k\in Q} n_{i,k}} = \frac{n_{i,j}}{n_i} \quad (3)$$

where n_i denotes the number of visits to state i. Note also that we slightly abuse notation in the above formula, by using the symbol Q, which usually refers to the set of states of the DFA, to also denote the set of states of the PMC.

In order to obtain a realization of the sequence Y of the states that $DFA_{\Sigma^*\cdot R}$ visits and the observed values $\hat{\pi}^{obs}_{i,j}$ as estimates for the transition probabilities, we can use an initial warm-up period during which a part of the stream is fed into the engine, the number of visits and transitions are counted and the transition probabilities are calculated, as per Equation (3) (line 2 in Algorithm 1).

ALGORITHM 1: Wayeb

Input: Stream S, pattern R, order m, maximum spread ms, forecasting threshold P_{fc}
Output: For each event $e \in S$, a forecast $I = (start, end)$

1 $DFA_{\Sigma^*\cdot R}$ = BuildDFA(R, m);
2 PMC^m_R = WarmUp(S, $DFA_{\Sigma^*\cdot R}$);
3 F_{table} = BuildForecastsTable(PMC^m_R, P_{fc}, ms);
4 *CurrentState* = 0;
5 *RunningForecasts* = ∅;
6 **repeat**
7 e = RetrieveNextEvent(S);
8 *CurrentState* = UpdateDFA($DFA_{\Sigma^*\cdot R}$, e);
9 **if** *CurrenState not final* **then**
10 $I = F_{table}$(*CurrentState*);
11 *RunningForecasts* = $I \cup$ *RunningForecasts*
12 **else**
13 UpdateStats(*RunningForecasts*);
14 *RunningForecasts* = ∅;
15 **end**
16 **until** *true*;

4.2 Building forecasts

After estimating the transition matrix, PMC^m_R is used in order to compute the waiting-time distributions for each non-final state. Based on these waiting-time distributions, we build the forecasts associated with each state (line 3). A forecast produced by Wayeb is in the form of an interval $I = (start, end)$. The meaning of this interval is the following: at each point, the DFA is in a certain state. Given this state, we forecast that the DFA will have reached its final state (and therefore the pattern fully matched) at some future point between $start$ and end, with probability at least P_{fc}. The calculation of this interval is done by using the waiting-time distribution that corresponds to each state and the threshold P_{fc} is set beforehand by the user.

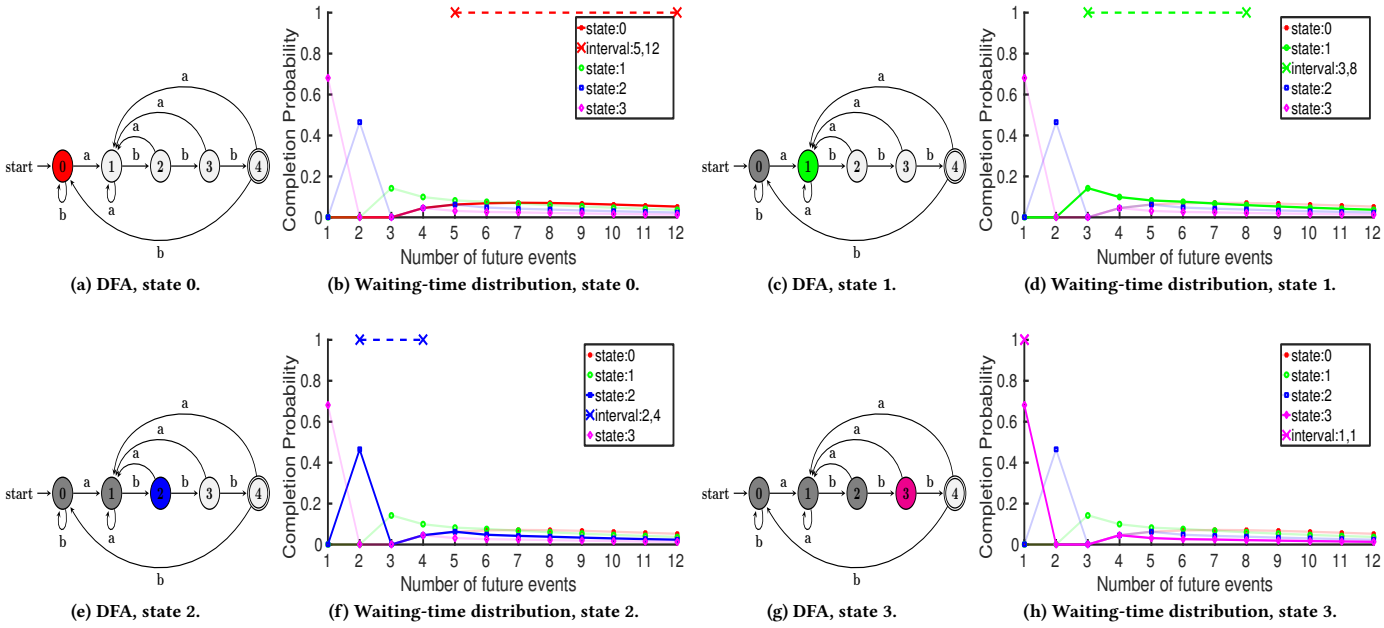
Each interval I that may be defined on the waiting-time distribution has an associated probability, given by:

$$P(I) = \sum_{n\in I} P(W_R(q) = n)$$

where we sum the probabilities of all points n that fall within I ($start \leq n \leq end$, where n is discrete). We define the set of intervals I_{fc} as:

$$I_{fc} = \{I : P(I) \geq P_{fc}\}$$

i.e., out of all possible intervals, I_{fc} contains those that have a probability above the user-defined threshold P_{fc}. Any one of the intervals



(a) DFA, state 0. (b) Waiting-time distribution, state 0. (c) DFA, state 1. (d) Waiting-time distribution, state 1.

(e) DFA, state 2. (f) Waiting-time distribution, state 2. (g) DFA, state 3. (h) Waiting-time distribution, state 3.

Figure 2: Example of how forecasts are produced. The pattern R is a sequential pattern $R = a \cdot b \cdot b \cdot b$ (one event of type a followed by three events of type b). $\Sigma = \{a, b\}$ (only two event types may be encountered, a and b) and $m = 1$. No maximum threshold for spread is set. $P_{fc} = 0.5$. For illustration purposes, the x axes stop at 12 future events.

in I_{fc} may be provided as a forecast. However, a lengthy interval (e.g., the whole domain of the distribution has probability 100% and therefore always belongs to I_{fc}) is less informative than a small one. Therefore, out of all the intervals in I_{fc}, we wish to provide the one that has the smallest length. We define the spread of an interval as:

$$spread(I) = end - start$$

The forecast is therefore given as:

$$I_{best} = \arg\min_{I \in I_{fc}} spread(I) \tag{4}$$

If more than one interval with the same smallest spread exist, then we choose the one with the highest probability. From each waiting-time distribution, we extract the best interval, as defined by Equation (4), using a single-pass algorithm that scans the distribution of each state only once. We may additionally require that the spread of the forecast interval is no greater than a specified maximum threshold *ms* (see relevant input argument in line 3 of Algorithm 1). In this case, however, it might not be possible to find an interval that satisfies both constraints

$$P(I) \geq P_{fc} \wedge spread(I) \leq ms$$

and the algorithm will return an empty interval.

An example of how forecasts are produced is shown in Figure 2. The pattern R is a simple sequential pattern $R = a \cdot b \cdot b \cdot b$ (one event of type a followed by three events of type b). Also $\Sigma = \{a, b\}$ (only two event types may be encountered, a and b) and $m = 1$. Therefore, the distributions are calculated based on the conditional probabilities $P(a|a)$, $P(a|b)$, $P(b|a)$ and $P(b|b)$. No maximum threshold for the spread has been set in this example. As shown in Figure 2a, the DFA has 5 states (0-4) and state 4 is the final state. When no event has arrived (or only b events have arrived), the DFA is in its start state. The waiting-time distribution for this state is shown in Figure 2b as the red curve. The other distributions are shown as well, but they are greyed out, indicating that only the red curve is "activated" in this state. If the user has set $P_{fc} = 0.5$, then the best interval that Wayeb can produce is the one shown above the distributions (red, dashed line), and this is $I = (5, 12)$. Notice that, as expected, according to the red distribution, it is impossible that the pattern is fully matched within the next three events (it is in the start state and needs to see at least 4 events). If an a event arrives, the DFA moves to its next state, state 1 (Figure 2c), and now another distribution is "activated" (green curve, Figure 2d). The best interval is now $I = (3, 8)$ and has a smaller spread. The arrival of a b event activates the blue distribution (Figure 2f) and this time an even smaller interval is produced, $I = (2, 4)$. If a second b event arrives, the magenta distribution is activated. This distribution has a peak above 0.5 which is the value of the threshold P_{fc} and this allows the engine to produce an interval with a single point $I = (1, 1)$. Essentially, Wayeb informs us that, with probability at least 50%, we will see a full match of the pattern in exactly 1 event from now.

Note that the calculation of the forecast intervals for each state needs to be performed only once, since for the same state it results always in the same interval being computed (assuming stationarity, as stated in Section 3). Therefore, the online forecasting system is again composed of a simple look-up-table (F_{table} in line 3 of Algorithm 1) and only memory operations are required.

4.3 Performance and quality metrics

There are three metrics that we report in order to assess Wayeb's performance and the quality of its forecasts:

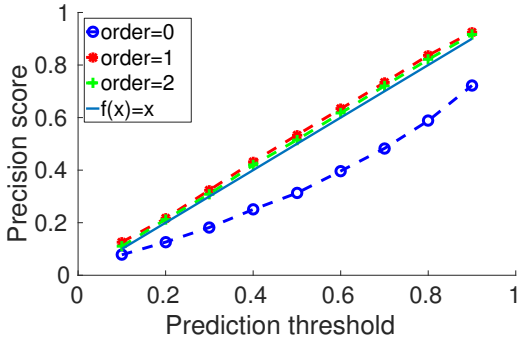
- *Precision* = $\frac{\#\ of\ correct\ forecasts}{\#\ of\ forecasts}$. At every new event arrival, the new state of the DFA is estimated (line 8 of Algorithm 1). If the new state is not a final state, a new forecast is retrieved from the look-up-table of forecasts (line 10). These forecasts are maintained in memory (line 11) until a full match is detected. Once a full match is detected, we can estimate which of the previously produced forecasts are satisfied, in the sense that the full match happened within the interval of a forecast (line 13). These are the correct forecasts. All forecasts are cleared from memory after a full match (line 14).
- *Spread* = $end - start$, as described in Section 4.2.
- *Distance* = $start - now$. This metric captures the distance between the time the forecast is made (*now*) and the earliest expected completion time of the pattern. Note that two intervals might have the same spread (e.g., $(2,2)$ and $(5,5)$ both have *Spread* equal to 0) but different distances (2 and 5, assuming $now = 0$).

Precision should be as high as possible. With respect to *Spread*, the intuition is that, the smaller it is, the more informative the interval. For example, in the extreme case where the interval is a single point, the engine can pinpoint the exact number of events that it will have to wait until a full match. On the other hand, the greater the *Distance*, the earlier a forecast is produced and therefore a wider margin for action is provided. Thus, "good" forecasts are those with high precision (ideally 1.0), low spread (ideally 0) and a distance that is as high as possible (ideal values depend on the pattern). These metrics may be calculated either as aggregates, gathering results from all states (in which case average values for *Spread* and *Distance* over all states are reported), or on a per-state basis, i.e., we can estimate the *Precision*, *Spread* and *Distance* of the forecasts produced only by a specific state of the DFA. We omit results for *Recall* (defined as percentage of detected events correctly predicted by at least one forecast), because *Recall* values are usually very high and not informative.

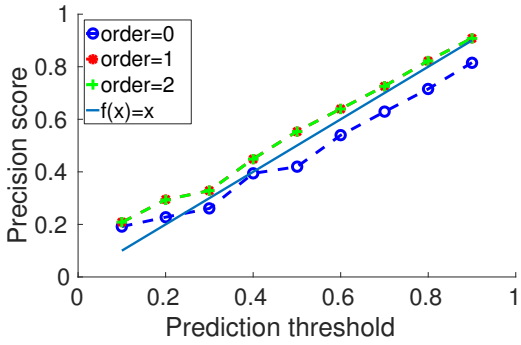
4.4 Validation tests with synthetic data

A set of tests was conducted with synthetically generated data for validation purposes. Streams were generated by a known Markov process and subsequently the engine was tested on these streams, for various patterns, forecast thresholds and orders. Figure 3 shows the aggregate (from all states) precision scores for two patterns, tested against a stream produced by a 1-order Markov process. The first pattern is the simple sequence $R = a \cdot b \cdot c$. The second pattern, $R = a \cdot (a+b)^* \cdot c$, is more complex and involves a *star closure* operation on the *union* of a and b, right after an a event and before a c event. For each pattern, three different values of the order m of the PMC were used (0, 1 and 2). The figures show how the engine behaves when the forecast threshold is increased and the order m of the PMC changes.

Note that the line $f(x) = x$ is also included in the figures, which acts as the baseline performance of the engine. If the Markov chain



(a) $R = a \cdot b \cdot c$.



(b) $R = a \cdot (a+b)^* \cdot c$.

Figure 3: Precision scores for synthetic data, produced by a 1-order Markov process, with $\Sigma = \{a, b, c\}$.

constructed for the pattern under test is indeed correct, then the precision score should lie above this line (or very close). As described in Section 4.2, each interval has a probability on the waiting-time distribution of at least $P_{fc} = x$. Therefore, if these waiting-time distributions are indeed correct, a percentage of at least P_{fc} of the intervals will be satisfied. This also means that the actual precision score might be significantly higher than the threshold in cases where the waiting-time distributions have high peaks. For example, in Figure 2h the single-point interval produced has a probability of $\approx 70\%$, hence $\approx 70\%$ of forecasts from that state will be satisfied, which is significantly higher than the 50% forecasting threshold.

For both of the tested patterns, when $m = 0$ (blue curves) the precision scores are below the baseline performance, indicating that a PMC without memory is unable to produce satisfactory forecasts for a 1-order stream. When m is increased to match the order of the generating process ($m = 1$, red curves), the precision score does indeed lie above the baseline. A further increase in the value of m does not seem to affect the precision score (in Figure 3b, the red and green curves for $m = 1$ and $m = 2$ respectively coincide completely and only the latter is visible).

In some cases, even if we use an incorrect order m, the precision score may be above the baseline or even above the line of the correct order m. This may happen because incorrect models may produce "pessimistic" intervals, with high spread and therefore implicitly take a bigger "chunk" out of the correct distributions. In practice,

however, the spread is constrained for informative forecasts, and thus models with incorrect order are insufficient.

5 EXPERIMENTAL RESULTS

We present results from experiments on real-world datasets from credit card fraud management and maritime monitoring.

5.1 Credit card fraud management

Unlike most academic and industrial work on fraud management, we performed an evaluation on a *real* dataset of credit card transactions, made available by Feedzai[1], our partner in the SPEEDD project[2]. Each event is a transaction accompanied by several arguments, such as the time of the transaction, the card ID, the amount of money spent, etc. There is also one boolean argument, indicating whether the transaction was labeled by a (human) analyst as being fraudulent or not. The original dataset is highly imbalanced. Only $\approx 0.2\%$ of the transactions are fraudulent. We created a summary of this original dataset, in which all fraudulent transactions were kept, but only some of the normal ones, so that the percentage of fraudulent transactions rises to $\approx 30\%$. The total number of transactions in this summary dataset was ≈ 1.5 million.

In order to be able to detect fraudulent transactions, companies use domain expert knowledge and machine learning techniques, so that they can extract a set of patterns, indicative of fraud. For our experiments, we used a set of fraud patterns provided by Feedzai, our partner in the SPEEDD project. We also employed the *partition-contiguity* selection strategy, where the ID of a card is used as the partition attribute. Upon the arrival of a new transaction event, the ID is checked and the event is pushed to the PMC run that is responsible for this ID or a new run is created, in case this transaction is the first one for this card.

In Figure 4, the results for the pattern *IncreasingAmounts* are presented, for three different values of the order m (1, 2 and 3), where we have set the maximum allowed spread at the value of 10. This pattern detects 8 consecutive transactions of a card in which the amount of money in a transaction is higher than the amount in the immediately previous transaction (for the same card ID), i.e., it attempts to detect sequences of transactions with increasing trends in their amounts. Since such direct relational constraints are not currently supported by our engine, a pre-processing step was necessary. During this step, each transaction is flagged as either being *Normal* or as one having an *IncreasingAmount* with respect to the immediately previous. Therefore, the pattern provided to Wayeb starts with one *Normal* transaction, followed by 7 transactions flagged as *IncreasingAmount*.

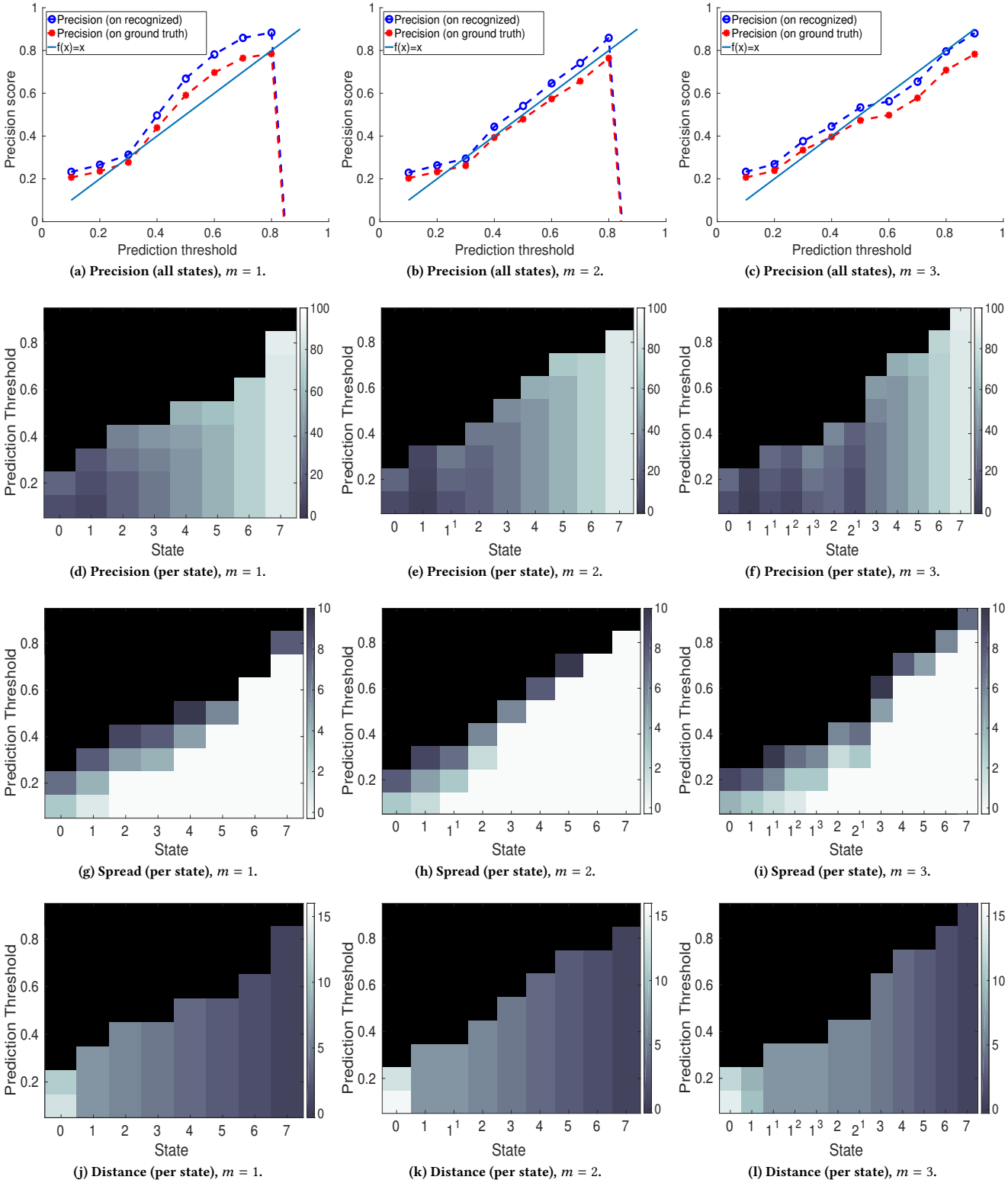
Since, in this dataset, there is ground truth available by fraud analysts, indicating whether a transaction was fraudulent or not, besides measuring precision with respect to the events detected by the PMC, we can also measure precision with respect to those fraud instances that were both detected and were actually marked as fraudulent. The red curves in the precision figures correspond to precision scores as measured when ground truth is taken into account. Note, however, that the dataset annotation does not contain information about the fraud type. This means that, when we detect a match of the *IncreasingAmounts* pattern and the ground truth informs us that the involved transactions are indeed fraudulent, there is no way to determine whether they are considered as fraudulent due to a trend of increasing amounts or to some other pattern. As a result, the red curves could be "optimistic".

For all three values of the order m, Wayeb can maintain a precision score that lies above the $f(x)=x$ line (Figures 4a and 4b) or is very close to it (Figure 4c), i.e., the produced forecasts, compared against the recognized matches (blue curves), satisfy the threshold constraint. However, when m=1 or m=2 and P_{fc}=0.9, Wayeb cannot find intervals whose probability is at the same time above this threshold and whose spread is below 10, and fails to produce any forecasts (the sudden drop in the curves indicates forecast unavailability). By increasing the order to m=3 and taking more past events into account (Figure 4c), Wayeb can handle this high forecast threshold (we will come back to this issue at the end of this section). As compared against the ground truth (red curves), the precision scores are lower. This precision discrepancy between scores estimated against recognized matches and scores estimated against ground truth is due to the fact that the fraud pattern is imperfect, i.e., there are cases with 8 consecutive transactions with *IncreasingAmount* which do not actually constitute fraud. It is interesting to note, though, that the shape of the red curves closely follows that of the blue ones, indicating that, by using a more accurate pattern, we would indeed be able to achieve ground truth precision closer to that of the blue curves for all values of P_{fc}.

The precision scores of Figures 4a, 4b and 4c are calculated by combining the forecasts produced by all states of the PMC. In order to better understand Wayeb's behavior, a look at the behavior of individual states could be more useful. Figures 4d – 4l depict image plots for various metrics against both the forecast threshold and the state of the PMC. The metrics shown are those of precision (on the recognized matches), spread and distance. We omit the plots for ground truth precision because they have the same shape as those for precision on recognized matches, but with lower values. In each such image plot the y axis corresponds to the various values of P_{fc}. The x axis corresponds to the states of the PMC. Each state has a unique integer identifier, starting from 0 (the start state). We group together states that are duplicates of each other, in cases where some states are ambiguous. For example, in Figure 4f, states 1^1, 1^2 and 1^3 are all duplicates of state 1. In this way, the x axis shows how advanced we are in the recognition process, when moving from one cluster of duplicates to the next. The black areas in these plots are "dead zones", meaning that, for the corresponding combinations of P_{fc} and state, Wayeb fails to produce forecasts (i.e., it cannot guarantee, according to the learned transition probabilities, that the forecast intervals will have at least P_{fc} probability of being satisfied). On the contrary, areas with light colors are "optimal", in the sense that they have high precision, low spread (the colorbar is inverted in the spread plots) and high distance in their respective plots.

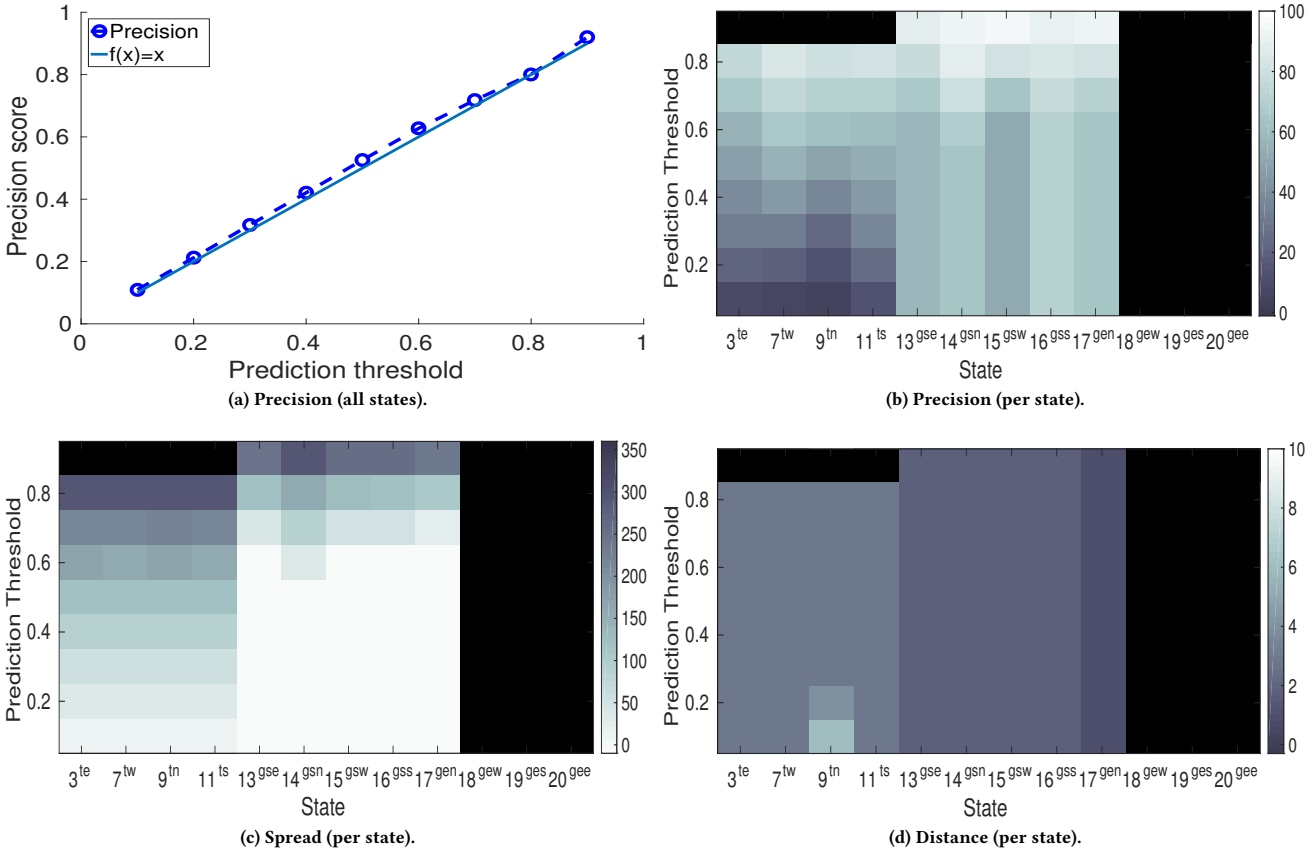
The precision plots (4d, 4e, 4f) show that the more advanced states of the PMC enter into such dead zones at higher forecast thresholds. Figures 4g, 4h and 4i show the spread of the forecast intervals. Two clearly demarcated zones emerge. One is the usual dead zone (black, top left). The other one (white, bottom right) corresponds to forecasts whose spread is 0, i.e., single point forecasts.

[1] https://feedzai.com/
[2] http://speedd-project.eu/



(a) **Precision (all states),** $m = 1$.
(b) **Precision (all states),** $m = 2$.
(c) **Precision (all states),** $m = 3$.
(d) **Precision (per state),** $m = 1$.
(e) **Precision (per state),** $m = 2$.
(f) **Precision (per state),** $m = 3$.
(g) **Spread (per state),** $m = 1$.
(h) **Spread (per state),** $m = 2$.
(i) **Spread (per state),** $m = 3$.
(j) **Distance (per state),** $m = 1$.
(k) **Distance (per state),** $m = 2$.
(l) **Distance (per state),** $m = 3$.

Figure 4: Results for the *IncreasingAmounts* **pattern, for** $m = 1$, $m = 2$ **and** $m = 3$, **and for maximum spread** $ms = 10$.



(a) Precision (all states).

(b) Precision (per state).

(c) Spread (per state).

(d) Distance (per state).

Figure 5: Results for the pattern *Turn* · *GapStart* · *GapEnd* · *Turn* **with** $m = 1$.

A common behavior for all states is that, as higher values of P_{fc} are set to the engine, the spread increases, i.e., each state attempts to satisfy the constraint of the forecast threshold by taking a longer interval from the waiting-time distribution. On the other hand, those states that are more advanced can maintain small spread values for a wider margin of P_{fc} values. For example, in Figure 4g, state 2 maintains a spread of 0 until P_{fc}=0.2 whereas state 5 hits this limit at around P_{fc}=0.5. Figures 4j, 4k and 4l show image plots for the distance metric. As can be seen, those regions that have high precision scores and low spread, also tend to have low distance. Therefore, there is a trade-off between these three metrics. For the case of m=1, good forecasts might be considered those of state 5, which can maintain a small spread until P_{fc}=0.5 and whose temporal distance is ≈ 3. By increasing m, one can get good forecasts that are more "satisfactory", at the cost of an increased size for the PMC (a discussion about this cost will be presented shortly). For example, when m=2, as in Figure 4h, state 5 can produce single point forecasts for higher values of P_{fc} (for 0.6 too).

As a final comment, we note that increasing the value of m does not necessarily imply higher precision scores. In fact, as shown in Figures 4a, 4b, 4c, the precision score might even decrease. This behavior is due to the fact that, in general, smaller values of m tend to produce more "pessimistic" intervals, with higher spread. For example, for P_{fc}=0.8, the precision score for m=1 (Figure 4a) is in fact higher than when m=2 (Figure 4b). In Figure 4g, we can see that, for m=1, the intervals of state 7 when P_{fc}=0.8 (the only state producing forecasts for this value of P_{fc}) have a high spread whereas the same state, when m=2, produces intervals with low spread (Figure 4h). Since "pessimistic", high-spread intervals take a bigger "chunk" out of a distribution, their precision scores end up being also higher. By increasing m, Wayeb can approximate the real waiting-time distributions more closely and therefore produce forecasts with lower spread that are closer to the specified threshold. Therefore, the accuracy curve (blue, dashed curve) starts to coincide with the $f(x)$=x line.

5.2 Maritime monitoring

Another real-world dataset against which Wayeb was tested came from the field of maritime monitoring. When sailing at sea, (most) vessels emit messages relaying information about their position, heading, speed, etc.: the so-called AIS (automatic identification system) messages. AIS messages may be processed in order to produce a compressed trajectory, consisting of critical points, i.e., important points that are only a summary of the initial trajectory, but allow for an accurate reconstruction [24]. The critical points of interest for our experiments are the following:

- *Turn*: when a vessel executes a turn.
- *GapStart*: when a vessel turns off its AIS equipment and stops transmitting its position.
- *GapEnd*: when a vessel turns on its AIS equipment back again (a *GapStart* must have preceded).

We used a dataset consisting of a stream of such critical points from ≈ 6.500 vessels, covering a 3 month period and spanning the Greek seas. Each critical point was enriched with information about whether it is headed towards the northern, eastern, southern or western direction. For example, each *Turn* event was converted to one of *TurnNorth*, *TurnEast*, *TurnSouth* or *TurnWest* events. We show results from a single vessel, with ≈ 50.000 events.

Figure 5 shows results for the pattern

$$Turn \cdot GapStart \cdot GapEnd \cdot Turn \tag{5}$$

where *Turn* is shorthand notation for

$$(TurnNorth + TurnEast + TurnSouth + TurnWest)$$

with + denoting the *OR* operator. Similarly for *GapStart* and *GapEnd*. With this pattern, we would like to detect a sequence of movements in which a vessel first turns (regardless of heading), then turns off its AIS equipment and subsequently re-appears by turning again. Communication gaps are important for maritime analysts because they often indicate an intention of hiding (e.g., in cases of illegal fishing in a protected area). The aggregate precision score (Figure 5a) is very close to the baseline performance. A look at the per-state plots reveals something interesting (Figures 5b, 5c, 5d). Note that, in order to avoid cluttering, we have removed duplicate states from the per-state plots. In addition, the superscript of each state in the x axis shows the last event seen when in that state. For example, the superscript te corresponds to *TurnEast*, tw to *TurnWest*, tn to *TurnNorth* and ts to *TurnSouth* (states 3, 7, 9 and 11 respectively). Similarly for *GapStart* for which superscripts start with gs (states 13–16) and for *GapEnd* (ge and states 17–20). These per-state plots show that there is a distinct "cluster" of states (13–17) which exhibit high precision scores for all values of P_{fc} (Figure 5b) and small spread for most values of P_{fc} (Figure 5c). Therefore, these states constitute what might be called "milestones" and a PMC can help in uncovering them. By closer inspection, it is revealed that states 13–16 are visited after the PMC has seen one of the *GapStart* events (we remind that *GapStart* is a disjunction of the four directional subcases). Moreover, *GapEnd* events are very likely to appear in the input stream right after a *GapStart* event, as expected, since during a communication gap (delimited by a *GapStart* and a *GapEnd*), a vessel does not emit any messages. State 17, which also has a similar behavior, is visited after a *GapEndNorth* event. Its high precision scores are due to the fact that, after a *GapEnd* event, a *Turn* event is very likely to appear. It differs from states 13–16 in its distance, as shown in Figure 5d, which is 1, whereas, for states 13–16, the distance is 2. On the other hand, states 18–20, which correspond to the other 3 *GapEnd* events, fail to produce any forecasts. The reason is that there are no such *GapEnd* events in the stream, i.e., whenever this vessel starts transmitting again after a *Gap*, it is always headed towards the northern direction.

Figure 6 shows results for the pattern

$$TurnNorth \cdot (TurnNorth + TurnEast)^* \cdot TurnSouth$$

This pattern is more complex since it involves a *star closure* operation on a nested *union* operation. It attempts to detect a rightward reverse of heading, in which a vessel is initially heading towards the north and subsequently starts a right turn until it ends up heading towards the south. Such patterns can be useful in detecting maneuvers of fishing vessels.

Figure 6 shows that a model with m=1 is unable to approximate well-enough the correct waiting-time distribution. Increasing the order to m=2 improves the precision score, but it still remains under the baseline performance. One could attempt to further increase the value of m, but this would substantially increase the cost of building the PMC. For $m = 1$, the generated PMC has ≈ 30 states. For $m = 2$, this number rises to ≈ 600 and the cost of creating an unambiguous DFA and then its corresponding PMC rises exponentially. When stationarity is assumed (as in our case) and the model does not need to be updated online, an expensive model can be tolerated.

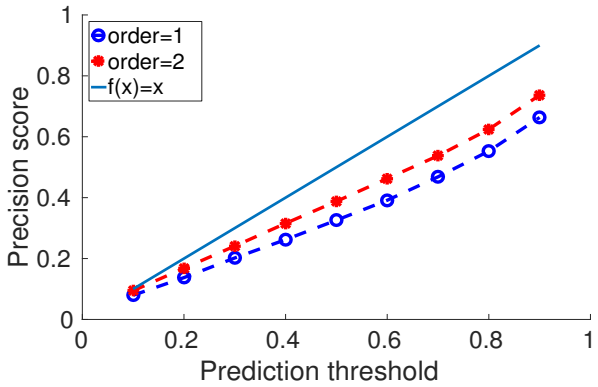


Figure 6: Results for the pattern $TurnNorth \cdot (TurnNorth + TurnEast)^* \cdot TurnSouth$.

5.3 Commentary on throughput

So far, we have focused on precision and quality metrics. For online event forecasting, throughput (defined as $\frac{\#\ of\ events\ consumed}{total\ execution\ time}$, where *execution time* refers to the **repeat** loop in Algorithm 1) is another important metric. We omit presenting detailed results about throughput, since Wayeb exhibits a steady behavior. For the maritime use case and the more complex heading reversal pattern, throughput is $\approx 1.2 \times 10^6$ *events/sec* and remains steady for both m=1 and m=2, whereas the event rate of the input stream is much lower. The experiments for the maritime use case were run on a 64-bit Debian machine, with Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz processors, and 16GB of memory. This high throughput number is due to the fact that the online operations of Wayeb consist mostly of memory operations (see Section 4). Even when the size of the PMC grows from ≈ 30 to ≈ 600, there is minimal overhead in accessing and maintaining the larger look-up-tables of the latter PMC. This independence from m also holds when multiple runs are employed, as in the credit card fraud use case (for the *partition − contiguity* selection strategy). Even in this case, only a single PMC is created (therefore, only one table for the DFA and one for the forecasts) and the different runs simply consult this PMC

through a reference to it. Throughput for the *IncreasingAmounts* fraud pattern is $\approx 1.2 \times 10^5$ *events/sec* (in total, 3 different patterns were tested), whereas the event rate at peak times reaches up to $\approx$ 1000 *events/sec*. Due to privacy reasons, experiments on the fraud dataset were run in Feedzai's premises and thus on different hardware: a 64-bit Ubuntu machine, with Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz processors and 32GB of memory. The multiple runs that need to be created, accessed and maintained with this dataset (on the contrary, for the maritime use case, only a single run is created) incur a significant increase in the execution time.

6 SUMMARY & FUTURE WORK

We presented Wayeb, a system that can produce online forecasts of event patterns. This system is not restricted to sequential patterns, but can handle patterns defined as regular expressions. It is also probabilistic and its forecasts can have guaranteed precision scores, if the input stream is generated by a Markov process. We have shown that it can provide useful forecasts even in real-world scenarios in which we do not know beforehand the statistical properties of the input stream. Moreover, the trade-off between precision score and the quality of the produced forecasts has been explored. Wayeb can also be used to uncover interesting probabilistic dependencies among the events involved in a pattern (pattern "milestones"), which can be informative in themselves or could possibly be used for optimization purposes in algorithms based on frequency statistics [15].

There are several directions for future research. One of them concerns relationality, i.e., our system should be able to handle directly constraints between the arguments of different events within a pattern. In this paper we focused on the *non-overlap* counting policy and the *contiguity* selection strategy. We have also implemented the *overlap* policy, but did not discuss it due to space limitations. With respect to the more flexible selection strategies (like *skip-till-any-match*), the usual way to deal with them is to clone runs of the automaton online, when appropriate. We could have followed a similar cloning approach as well and produce forecasts for each run. However, it is doubtful whether individual forecasts made by a multitude (possibly hundreds) of concurrently existing runs would be useful to a user. Some form of aggregate forecasting (e.g., number of full matches expected within the next N events) could be more informative. We intend to pursue this line of research. Another useful functionality would be that of assessing whether the model should be updated online, once we drop the stationarity assumption, and how this could be done efficiently.

ACKNOWLEDGMENTS

This work was funded partly by the EU H2020 datACRON and partly by the EU FP7 SPEEDD projects. We wish to thank Feedzai, our partner in SPEEDD, for providing access to the credit card fraud data and Evangelos Michelioudakis for running the experiments in the premises of Feedzai.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules Between Sets of Items in Large Databases. In *ACM SIGMOD*.
[2] Chung-Wen Cho, Yi-Hung Wu, Show-Jane Yen, Ying Zheng, and Arbee L. P. Chen. 2010. On-line rule matching for event prediction. *The VLDB Journal* (2010).
[3] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. 2007. *Algorithms on strings.* Cambridge Univ. Press.
[4] Carlotta Domeniconi, Chang-shing Perng, Ricardo Vilalta, and Sheng Ma. 2002. A Classification Approach for Prediction of Target Events in Temporal Sequences. In *Principles of Data Mining and Knowledge Discovery*. Springer.
[5] Christophe Dousson and Pierre Le Maigat. 2007. Chronicle Recognition Improvement Using Temporal Focusing and Hierarchization.. In *IJCAI*, Vol. 7.
[6] Yagil Engel and Opher Etzion. 2011. Towards Proactive Event-driven Computing. In *ACM DEBS*.
[7] Opher Etzion. 2016. Proactive Computing: Changing the Future. https://www.rtinsights.com/proactive-computing-prescriptive-analytics-special-report/. (2016). [Online; accessed 23-February-2017].
[8] Lina Fahed, Armelle Brun, and Anne Boyer. 2014. Efficient Discovery of Episode Rules with a Minimal Antecedent and a Distant Consequent. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management*. Springer.
[9] James C Fu and W. Y. Wendy Lou. 2003. *Distribution theory of runs and patterns and its applications: a finite Markov chain imbedding approach.* World Scientific.
[10] Lajos Jenő Fülöp, Árpád Beszédes, Gabriella Tóth, Hunor Demeter, László Vidács, and Lóránt Farkas. 2012. Predictive Complex Event Processing: A Conceptual Framework for Combining Complex Event Processing and Predictive Analytics. In *Proceedings of the Fifth Balkan Conference in Informatics*. ACM.
[11] Malik Ghallab. 1996. On chronicles: Representation, on-line recognition and learning. In *KR*.
[12] Asela Gunawardana, Christopher Meek, and Puyang Xu. 2011. A Model for Temporal Dependencies in Event Streams. In *Advances in Neural Information Processing Systems 24*. Curran Associates, Inc.
[13] Dan Gusfield. 1997. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge Univ. Press.
[14] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2007. *Introduction to automata theory, languages, and computation.* Pearson/Addison Wesley.
[15] Ilya Kolchinsky, Izchak Sharfman, and Assaf Schuster. 2015. Lazy Evaluation Methods for Detecting Complex Events. In *DEBS*.
[16] Srivatsan Laxman, Vikram Tankasali, and Ryen W. White. 2008. Stream Prediction Using a Generative Model Based on Frequent Episodes in Event Sequences. In *ACM SIGKDD*.
[17] Manuel E. Lladser, M. D. Betterton, and Rob Knight. 2007. Multiple pattern matching: a Markov chain approach. *Journal of Mathematical Biology* (2007).
[18] M Lothaire. 2005. *Applied combinatorics on words.* Cambridge Univ. Press.
[19] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. 1997. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery* (1997).
[20] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. 2015. *Introduction to time series analysis and forecasting.* Wiley.
[21] Vinod Muthusamy, Haifeng Liu, and Hans-Arno Jacobsen. 2010. Predictive Publish/Subscribe Matching. In *DEBS*. ACM.
[22] Pierre Nicodème, Bruno Salvy, and Philippe Flajolet. 2002. Motif statistics. *Theoretical Computer Science* (2002).
[23] Grégory Nuel. 2008. Pattern Markov Chains: Optimal Markov Chain Embedding through Deterministic Finite Automata. *Journal of Applied Probability* (2008).
[24] Kostas Patroumpas, Elias Alevizos, Alexander Artikis, Marios Vodas, Nikos Pelekis, and Yannis Theodoridis. 2016. Online event recognition from moving vessel trajectories. *GeoInformatica* (2016).
[25] R. Vilalta and Sheng Ma. 2002. Predicting rare events in temporal domains. In *ICDM*.
[26] Gary M. Weiss and Haym Hirsh. 1998. Learning to Predict Rare Events in Event Sequences.. In *KDD*.
[27] Z. Xing, J. Pei, G. Dong, and P. Yu. 2008. Mining Sequence Classifiers for Early Prediction. In *SIAM Conference on Data Mining*.
[28] Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2014. On Complexity and Optimization of Expensive Queries in Complex Event Processing. In *ACM SIGMOD*.
[29] Cheng Zhou, Boris Cule, and Bart Goethals. 2015. A pattern based predictor for event streams. *Expert Systems with Applications* (2015).