UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Anton Yeshchenko

An Eye into the Future: Leveraging A-Priori Knowledge in Predictive Business Process Monitoring

Master's Thesis (20 ECTS)

Supervisor: Fabrizio Maria Maggi
Supervisor: Chiara Di Francescomarino

Tartu 2017

Abstract. Predictive business process monitoring aims at leveraging past process execution data to predict how ongoing (uncompleted) process executions will unfold up to their completion. Nevertheless, cases exist in which, together with past execution data, some additional knowledge (a-priori knowledge) about how a process execution will develop in the future is available. This knowledge about the future can be leveraged for improving the quality of the predictions of events that are currently unknown. In the thesis, we present two techniques - based on Recurrent Neural Networks with Long Short-Term Memory (LSTM) cells - able to leverage knowledge about the structure of the process execution traces as well as a-priori knowledge about how they will unfold in the future for predicting the sequence of future activities of ongoing process executions. The results obtained by applying these techniques on six real-life logs show an improvement in terms of accuracy over a plain LSTM-based baseline.

Keywords: Predictive Process Monitoring, Recurrent Neural Networks, Linear Temporal Logic, A-priori Knowledge, Process mining

Table of Contents

1 Introduction

We have come to the world where success is highly correlated with the ability for businesses to provide superior delivery of services compared to their rivals. Competition made companies to seek for better management and organization of their work-flow. Furthermore, technologies also evolved, enabling the use of advanced methods for analysis that was not possible before. To assist business needs, Process Mining field has evolved. It designs techniques to analyze performance of the company from event logs.

Predictive monitoring [23] is a maturing subfield of Process Mining, that deals with online prediction of process outcomes. Predicted outcomes may be of different nature, that are defined classes, degree of compliance with a measure, or achieving objective.

An example would be predicting the compliance of a process execution in insurance company, given annotated log. Given uncompleted traces of the claims processes, the goal would be to predict if customers will have the decisions on their claim on time. These predictions are generally based on: (i) the *sequence of activities* already executed in the case; (ii) the *timestamp* indicating when each activity in the case was executed; and (iii) the *values of data attributes* after each execution of an activity in the case.

Predictive monitoring undergoes rapid development of algorithms and techniques focused at predicting an outcome (class label of the process) as well as predicting ongoing process execution (text activity to be performed). Also, to enhance the decision making of business processes, systems that can give recommendations based on current process execution, given log, and some performance measures are built.

Problems of this kind could be viewed as standard machine learning problems. Given event log consisting of traces as a set of feature vectors in classic machine learning, we aim to predict labels (classification), or sequence of next events (sequence to sequence learning) of future process executions. But the real world process mining problems are daunting to the existing machine learning theory, as there are no methods that adapt easily to all additional requirements.

Many sophisticated methods[14,27,33] are currently available for prediction of business processes, but none of them account for the prior knowledge that can be additionally available on execution time.

What motivates this thesis is the surmise that past event logs, or more in general knowledge about the past, is not the only important source of knowledge that can be leveraged to make predictions. In many real life situations, cases exist in which, together with past execution data, some case-specific additional knowledge (a-priori knowledge) about the future is available and can be leveraged for improving the predictive power of a predictive process monitoring technique. Indeed, this additional a-priori knowledge is what characterizes the future context of execution of the process that will affect the development of the currently running cases. Think for instance to the temporary unavailability of a surgery room which may delay or even rule out the possibility of executing certain activities in a patient treatment process. While it is impractical to retrain the predictive algorithms to take into consideration this additional knowledge every time it becomes available, it is also reasonable to assume that considering it in some way would improve the accuracy of the predictions on an ongoing case.

While this observation seems plausible and a-priori knowledge appears to have the potential of improving predictions, its concrete realization poses interesting challenges: a first challenge is the provision of actual techniques able to leverage on past event logs and a-priori knowledge for predicting the sequence of future activities in ongoing traces. A second challenge is the actual verification that these techniques can show an improvement in terms of accuracy on real life logs over a plain baseline that leverages only knowledge from past events.

Therefore, our work is concerned at following closely the research question:

RQ: How can prior knowledge about ongoing process be used to boost accuracy of predictions of trends of activities?

In light of this motivation, in Section 5, we provide two techniques based on Recurrent Neural Networks with Long Short-Term Memory (LSTM) cells [18] able to leverage a-priori knowledge about process executions for predicting the sequence of future activities of an ongoing case. The proposed algorithms are opportunely tailored in a way that the a-priori knowledge is not taken into account for training the predictor. In this way, the a-priori knowledge can be

changed on-the-fly at prediction time without the need to retrain the predictive algorithms. In particular, we introduce:

- a NOCYCLE technique which is able to leverage knowledge about the structure of the process execution traces, and in particular about the presence of repetitions of sequences (i.e., cycles), to improve a plain LSTM-based baseline so that it does not fall into a local minimum, a phenomenon already hinted in [33] but not yet solved;
- an A-PRIORI technique which takes into account a-priori knowledge together with the knowledge that comes from historical data.

In Section 6, we present a wide experimentation carried out using six real-life logs and aimed at investigating whether the proposed algorithms increase the accuracy of the predictions. The outcome of our experiments is that the application of these techniques provides an improvement up to 50% in terms of prediction accuracy over the baseline. In addition to the core part (Sections 5 and 6), the paper contains an introduction to some background notions (Section 2), a detailed illustration of the research problem (Section 4), related work (Section 3) and concluding remarks (Section 7).

2 Background

In this section, we report the background concepts useful for understanding the remainder of the thesis.

2.1 Event Logs and Traces

Most of the huge enterprises and middle sized companies have a lot of processes being executed. With the rapid development of database technology and inexpensiveness of data warehouses saving all the transactional data in the company became a must. Most of this data is about processes that are happening in the company. Those range from core processes such as "order to cash" and "issue to resolution" to additional such as managerial processes. The nature of this information is processlike (sequential) so they are usually being recorded in the format of logs.

An event log is a set of traces, each representing the execution of a process (case instance). Each trace consists of a sequence of

activities, each referring to the execution of an activity in a finite activity set A.

Definition 1 (Trace, Event Log). *A trace* $\sigma = \langle a_1, a_2, ...a_n \rangle \in A^*$ *over* A *is a sequence of activities. An event log* $L \in \mathcal{B}(A)$ *is a multi-set of traces over the activity set* A.

A *prefix of length* k of a trace $\sigma = \langle a_1, a_2, ...a_n \rangle \in A^*$, is a trace $p_k(\sigma) = \langle a_1, a_2, ...a_k \rangle \in A^*$ where $k \leq n$; the *suffix of the prefix of length* k is defined as the remaining part of σ, that is, $s_k(\sigma) = \langle a_k + 1, a_k + 2, ...a_n \rangle \in A^*$. For example, the prefix of length 3 of $\langle a, c, r, f, s, p \rangle$ is $\langle a, c, r \rangle$, while the suffix of this prefix is $\langle f, s, p \rangle$.

A *cycle* in a trace $\sigma \in A^*$ is a sequence of activities repeated at least twice in σ (with adjacent repetitions). For example, trace $\langle a, b, a, b, a, b, c, d, e, f, g, e, f, g, c, d \rangle$ contains two cycles: $\langle a, b \rangle$ (3 repetitions) and $\langle e, f, g \rangle$ (2 repetitions).

2.2 Process mining

Process mining is a business process management technique that enables the analysis of processes based on the event logs. Main idea lies at improving business processes based on analysis of the logs as sequential, continuous processes. That what makes it different from the methods established in data mining and business intelligence which treat the data as set of distinct events to gather insights.

As a rule the methods of process mining applied mostly to the processes that do not have formal description or have consistent deviations from the predicted in the system.

The three main activities performed under the process mining field are:

- **Process analysis (Discovery)** encompasses all methods to discover insights from the past process executions.
- **Conformance analysis** is to take existent formalized model check it against real log
- **Process enhancement** helps to use log in order to enhance current executions of the processes

2.3 Predictive monitoring

Predictive monitoring is an advancing-sub field of Process Mining. As a a sub field of Process Mining, its main focus also lies in ex-

ploring and exploiting the process logs. More specifically, it deals with online predictions of process outcomes. Predicted outcomes of interest can be compliance of with a measure, achieving objective, or classifications. Classification is giving a label to the uncompleted trace.

Many algorithms are applied to mentioned problems, these include classic machine learning algorithms of classification and clustering. More complex concepts are also used, such as Markov models [21], time series, and deep learning[33,14,37] approaches.

In order to successfully apply these algorithms for aforementioned problems, one usually needs to have solid understanding on the process log nature. The aspects that can come into play, are the data types, stationarity, or heteroscedasticity. More complications can be brought if log contains unstructured content such as textual comments.

A log is a set of traces, each containing events with its even class, ID, name, and usually time stamp and n-dimensional payload vector. Events correspond to particular cases in the process. Early process mining approaches tend to ignore the data payload, taking into consideration only the control-flow (sequence of activities). Some of the approaches do the opposite, that is consider data, without control-flow [3,4,28]

In that case one can say that the traces were considered to be a simple sequences. Moreover, methods introduced usually designed to work in offline fashion, exploring completed cases. But, to make more accurate, mature, online predictions one should abandon neither payload nor control-flow information. That's when the Complex Symbolic Sequences as methods of encoding of a log are introduced.

2.4 Complex Symbolic Sequences

Every machine learning problem needs the data represented in a particular way, and algorithms adapted to the encoded data. For classical predictions of next steps in the sentence having bounded space of variants at each step the Simple Symbolic Sequence encoding is used. It is an encoding representing the sequence in a vector or a array of symbols.

Simple Symbolic Sequences for log encoding can be used with a range of methods inherited from a massive number of works from

machine learning and statistics such as concerning problems from health-informatics, or anomaly detection in Unix access systems. determining fraud users by its query log sequences. [38] That is because the simple symbolic sequence encoding is simply an ordered list sampled out of some given alphabet.

With the use of Simple Symbolic Sequences, it was relatively simple to apply methods, feature based classifications, sequence distance based, support vector machines, or model based classification [38].

Simple Symbolic Sequences have limitations that prevent getting best results of their use. That is because simple symbolic sequences discard most of the rich information present in process logs, that is not limited to control-flow. An example would be the log in the big fast food chain, where the process about their employee's actions is daily recorded into the log. In Table 1, you can see that log of this nature contains static features, control-flow components and number of dynamic features. Static feature include ID of the employee, and his age. Dynamic features are the t_i that is the time spent working, rID_i is an ID of the restaurant that employee worked in on that day, rev_i money he earned for the company.

This log is rich with information additional information, and in order to work with it, one would want to capture as much information as possible.

On contrary to Simple Symbolic Sequences, the Complex Symbolic Sequences are introduced and we can capture much more information about log. Complex Symbolic Sequence is basically ordered list of vectors, where each vector is a subset of some alphabet inferred from the log.

	Age	ID	t_1	..	t_n	rID_1	..	rID_n	rev_1	..	rev_n
tr1	20	00123	28800	..	24840	11	..	12	120$	..	112$
tr2	24	00023	16890	..	23460	12	..	12	80$	..	111$
tr3	18	01129	23804	..	17897	11	..	11	340$	..	23$
tr4	21	00772	40034	..	24564	12	..	12	130$	..	2$

Table 1: Example of Complex Symbolic Sequence encoding

When Complex Symbolic Sequences are used instead of Simple Symbolic Sequences complexity space of the prediction problem

grows substantially. So it suggests to use new and more complex approaches to work with them.

Some of the base lines of working with complex symbolic sequences use index-based encodings, and index latest payload encodings mentioned in following paper [21].

2.5 Structured and unstructured data

There are few distinct types of data, that can be stored in the event. Simple, structured types include time-stamps, numbers, and nominal values. If the log contains only these types it is called a log with structured data.

On the other hand, the logs can contain text information such as comments or emails, bringing up the complexity level of the problem. Text mining methods in conjunction with sequence classification should be used to bring down the problem [34].

As for example mentioned above, it might be a daily comment from manager of the employees in the fast-food chain.

Also, as for example of the issue to resolution process, textual comments on performance of the employee are added to events. In the repair agency, the comments can be added with registering of the issue by customer (describing the initial assessment of the issue), and later in repair process by any involved party, that are repairmen, managers, or customers. Also comments can be some content of the emails what are associated with a business process in the company.

With this knowledge about types of business processes it is easier to understand the limitations of the current state of the art methods to be described in the related work section.

2.6 Hidden Markov Models

Hidden Markov Models (HMMs) are one of the most used probabilistic models. These models belong to the family of directed probabilistic graphical models.

As for the model, random variables are divided into two sets, namely a set seen outside, an *observations* set $X = X_1, ..., X_i, ..., X_n$, and a hidden sequence (that is considered to be a first order Markov chain) of *states* $Y = Y_1, ..., Y_i, ..., Y_n$.

The model is built by four sets of parameters.

$P(Y_1 = c_k \mid Y_0 = start)$,	initial probability
$P(Y_i = c_k \mid Y_{i-1} = c_l)$	transition probability
$P(Y_i = stop \mid Y_{i-1} = c_k)$	final probability
$P(X_i = w_j \mid Y_i = c_k)$	emission probability

The observation set is the one that is seen on the outside. The usual usage of HMMs is having an HMM defined by parameters above to predict the sequence of states.

In order to predict the sequence the Viterbi algorithm is used. It is a perfect example of a dynamic programming algorithm to find *max-product* of probabilities over all pace of sequences.

2.7 Conditional Random Field

The HMM model described above is a generative model, that is used to model the joint distribution of the outcome and the features $P(Y, X)$. There is another version of sequential models that is discriminative. It is called Conditional Random Field. It is used to build a conditional probability $P(Y|X)$ of the outcome.

CRF is a random field, in which the output y is conditioned on input x. The conditional probability takes form

$$P(Y|X) = 1/Z(X) * \exp(\sum_{m=1}^{M} (\lambda_m * f_m(y_n, y_{n-1}, x_n)))$$

where the $Z(X)$ normalization function of the form

$$Z(X) = \sum_{y} \exp(\sum_{m=1}^{M} (\lambda_m * f_m(y_n, y_{n-1}, x_n)))$$

The main advantage of this model is relaxation of condition of observed data with labels.

2.8 RNNs and LSTM

Artificial Neural Networks (or just Neural Networks, NNs) are a well known class of discriminative models. In classification tasks, they are used to model the probability of a given input to belong to a certain

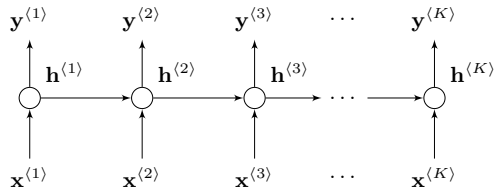

Fig. 1: Recurrent Neural Network

class, given some features of the input. We can describe them in mathematical terms as follows:

$$p(\mathbf{y}|\mathbf{x}) = f_{NN}(\mathbf{x};\theta). \tag{1}$$

In (1), $\mathbf{x}$ is the feature vector that represents the input, $\mathbf{y}$ is a random variable representing the output class labels, f_{NN} is the function modeled by the neural network, and θ is the set of parameters of such a function to be learnt during the training phase.

Recurrent Neural Networks (RNNs, see Fig. 1) are a subclass of Neural Networks. We illustrate them with the help of an example in which the classification task concerns in assigning the correct part of speech – noun, verb, adjective, etc. – to words. If we take the word *"file"* in isolation, it can be both a noun and a verb. Nonetheless, this ambiguity disappears when we consider it in an actual sentence. Therefore, in the sentence *"I have to file a complain"* it acts as a verb, while in the sentence *"I need you to share that file with me"* it acts as a noun.

This simple example shows that for some tasks the classification at a certain time-step t depends not only on the current input (i.e., *"file"*) but also on the input (i.e., the part of the sequence) seen so far. The tasks that share this characteristic are said to be *recurrent*. Natural Language tasks are a typical example of recurrent phenomena.

In mathematical terms, let us write $\mathbf{x}^{\langle 1\rangle}, ..., \mathbf{x}^{\langle K\rangle}$ to indicate an input sequence of K time-steps, represented by the superscript between angle brackets. In this way, at each time-step t, the conditional probability of a given input to belong to a certain class is described by

$$p(\mathbf{y}^{\langle y\rangle}|\mathbf{x}^{\langle t\rangle}, ..., \mathbf{x}^{\langle 1\rangle}) = f_{RNN}(\mathbf{x}^{\langle 1\rangle}, ..., \mathbf{x}^{\langle t\rangle};\theta). \tag{2}$$

RNNs have been proven to be extremely appropriate for modeling sequential data (see [16]). As shown in Fig. 1, they typically leverage recurrent functions in their hidden layers, which are, in turn, composed of hidden states. Let $\mathbf{h}^{\langle t \rangle}$, with

$$\mathbf{h}^{\langle t \rangle} = h(\mathbf{x}^{\langle t \rangle}, \mathbf{h}^{\langle t-1 \rangle}; \theta_h); \tag{3}$$

be the activation of the hidden state at the t-th time-step. h is a so-called *cell function*, parameterized over a set of parameters θ_h to be learnt during the training, and accepting as inputs the current input $\mathbf{x}^{\langle t \rangle}$ and its value at the previous time-step $\mathbf{h}^{\langle t-1 \rangle}$. The activation of the hidden state is then mapped (using a linear map) into a continuous vector of the same size as the number of output classes. All the elements in such a vector are greater than zero and their sum is equal to one. Therefore, this vector can be seen as a probability distribution over the output space. All these constraints can be easily achieved specifying the generic equation (2) by means of a softmax function:

$$p(\mathbf{y}^{\langle y \rangle} | \mathbf{x}^{\langle t \rangle}, ..., \mathbf{x}^{\langle 1 \rangle}) = softmax(\mathbf{W}\mathbf{h}^{\langle t \rangle} + \mathbf{b}); \tag{4}$$

where the weight matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$ are parameters to be learnt during the training phase.

Among the different cell functions h (see equation (3)) explored in literature, Long Short-Term Memory (LSTM) [18] shows a significant ability to maintain the memory of its input across long time spans. This property makes them extremely suitable to be used in RNNs that have to deal with input sequences with complex long-term dependencies such as the ones we consider in this paper.

2.9 RNNs with LSTM for Predictive Process Monitoring

In order to provide predictions on the suffix of a given prefix (of a running case), state-of-the-art approaches for predictive process monitoring use RNNs with LSTM cells. The most recent and performing approach in this field [33] relies on an encoding of activity sequences that combine features related to the activities in the sequence (the so called *one-hot encoding*) and features related to the time characterizing these activities. Given the set $A = \{a_{1_A}, \dots a_{m_A}\}$ of all possible activities, an ordering function $idx : A \to \{1, \dots, |A|\} \subseteq \mathbb{N}$

is defined on it, such that $a_{i_A} <> a_{j_A}$ if and only if $i_A <> j_A$, i.e., two activities have the same A-index if and only if they are the same activity. For instance, if $A = \{a, b, c\}$, we have $idx : A \rightarrow \{1, 2, 3\}$ and $idx(a) = 1$, $idx(b) = 2$ and $idx(c) = 3$. Each activity $a_i \in \sigma$ is encoded as a vector (A_i) of length $|A| + 3$ such that the first $|A|$ features are all set to 0, except the one occurring at the index of the current activity $idx(a_i)$, which is set to 1. The last three features of the vector pertain to time: the first one relates to the time increase with respect to the previous activity, the second reports the time since midnight (to distinguish between working and night time), and the last one refers to the time since the beginning of the week.

A trace is encoded by composing the vectors obtained from all activities in the trace into a matrix. During the training phase, the encoded traces are used for building the LSTM model. During the testing phase, a (one-hot encoded) prefix of a running case is used to query the learned model, which returns the predicted suffix by running an inference algorithm. Algorithm 1 reports the inference algorithm introduced in [33] and based on RNN with LSTM cells for predicting the suffix of a given prefix $p_k(\sigma)$ of length k. The algorithm takes as input the prefix $p_k(\sigma)$, the LSTM model $lstm$ and a maximum number of iterations max and returns as output the complete trace (the prefix and the predicted suffix). First, the prefix $p_k(\sigma)$ is encoded by using the one-hot encoding (line 5). The resulting matrix is then used for feeding the LSTM model and getting the probability distribution over different possible symbols that can occur in the next position of the trace (line 6). The symbol with the highest probability is hence selected from the ranked probabilities (line 7). Then, a new trace is obtained by concatenating the current prefix with the new predicted symbol (line 8). In order to predict the second activity, the one-hot encoding of the new prefix is computed and used to recursively feed the network. The procedure is iterated until the predicted symbol is the end symbol or a maximum number of iterations max is reached (line 10).

2.10 Linear Temporal Logic

In our approach, the a-priori knowledge that describes how a running case will develop in the future is formulated in terms of Linear

Algorithm 1 Inference algorithm for predicting the suffix of $p_k(\sigma)$

```
1: function PredictSuffix(p_k(σ), lstm, max)
2:     k = 0
3:     trace = p_k(σ)
4:     do
5:         trace_encoded = encode(trace)
6:         next_symbol_probs = predictNextSymbols(lstm, trace_encoded)
7:         next_symbol = getSymbol(next_symbol_prob, trace_encoded)
8:         trace = trace · next_symbol
9:         k = k + 1
10:    while (not next_symbol == end_symbol) and (k <max)
11:    return trace
12: end function
```

Temporal Logic (LTL) rules [26]. LTL is a modal logic with modalities devoted to describe time aspects. Classically, LTL is defined for infinite traces. However, to describe the characteristics of a business process, we use a variant of LTL defined for finite traces (since business processes are supposed to complete eventually). We assume that activities occurring during the process execution fall into the set of atomic propositions. LTL rules are constructed from these atoms by applying the temporal operators $\bigcirc$ (next), $\Diamond$ (future), $\Box$ (globally), and $\sqcup$ (until) in addition to the usual boolean connectives. Given a formula φ, $\bigcirc\varphi$ means that the next time instant exists and φ is true in the next time instant (strong next). $\Diamond\varphi$ indicates that φ is true sometimes in the future. $\Box\varphi$ means that φ is true always in the future. $\varphi \sqcup \psi$ indicates that φ has to hold at least until ψ holds and ψ must hold in the current or in a future time instant.

2.11 Beam search

Beam search is a heuristic search algorithm, based on the principle of exploring the graph of possible solutions by the most promising node. It is built on the assumption of huge search space. It orders all partial solutions with some heuristic measure keeping only small portion of the most promising ones, expanding them to get the complete solution.

There are few variations of this algorithm, from which two will be thoroughly explained here.

Best-first variation of Beam search Best-first search explores the solution space following the most probable solution first. The algorithm exploits a tree structure, in which each node maintains its state as open and closed. The algorithm assumes that all nodes are open at the initialization, and after visiting the nodes marks them as closed.

The pseudo code for the algorithm is following:

Algorithm 2 Best first search

```
1: define node of the graph, OPEN, to be starting node s
2: if LIST is empty then
3:     failure
4: end if
5: Take the best node from the list (call it n), and make it CLOSED
6: Expand node n
7: if any of the successor nodes are the solution then
8:     success, return the solution traced back
9: end if
10: for all node of the successors do
11:     apply evaluation function f to the node
12:     if node is new (not CLOSED) mark as OPEN
13: end for
```

Different evaluation function can be used for the task. The decision is made upon the specifics of the problem, and it directly influences the effectiveness of the algorithm.

Breath-first variation of Beam search Breath-first search begins with an empty node and explores the search space in the breath-wise order.

This type of beam search operates using few priority queues instead of tree structures as in previous approach.

Here is the pseudo code for the algorithm (all elements in priority queues are stores with priority based on some evaluation function):

Both of these algorithms use the same tactic by cutting the space to the most probable and better performing prospective solutions. These algorithms are different, and depending on the nature of the problem given can have very different performance.

Beam search is used in many areas such as machine translation[20].

Algorithm 3 Breath-first Beam search

```
1: define two empty priority queues, s1 and s2
2: add the first element to s1
3: while solution is not found or the search space is not empty do
4:     while s1 is not empty do
5:         get element elem from s1
6:         if solution found then
7:             success
8:         end if
9:         expands elem and write beamsize-elements to s2
10:    end while
11:    s2 equals first part of s1 with the size of beamsize
12: end while
```

3 Related Work

Our research question:

RQ: How can prior knowledge about ongoing process be used to boost accuracy of predictions of trends of activities?

Basically, the research question concerns predicting next events in the sequence, focusing on control-flow activity for next predicted event.

In the following section the techniques and algorithms used in recent papers that are considered as a state of the art to the problems of interest will be described.

Firstly, the historic overview of early approaches will be given. Secondly, the state of the art methods based on Complex Symbolic Sequences Encodings will be described. Finally, the overview of the algorithms for predictions will be given, and it will be specified that there are no algorithms yet, that encompass the online prediction of process outcomes with A-priori knowledge.

3.1 Early approaches

In the early applications of business process monitoring, few mathematical models were brought to the field. The first were Petri nets [36]. Petri nets prove to be a very robust tool for a range of problems in process mining. They are used for conformance checking, process discovery, and process monitoring. They have few important properties such as having formal semantics. Also, they are simplistic models,

and have graphical nature so they are expressive. Their properties are mathematically investigated.

Later, the limitations for application of Petri nets in predictive monitoring were amplified, by finding new methods for the problems.

Kang and Jung in the paper [19] propose a new approach for monitoring the progress of ongoing processes and predicting probable performances and routes. Their contributions based on utilizing the Formal Concept Analysis (FCA) for monitoring business processes. They propose an alternative approach to Petri nets for visualizing the ongoing processes and predicting the future evolvings introducing concept lattice and reachability lattice.

Also to the first group of works we can associate the paper [4], the authors of which present a set of approaches in which annotated transition systems, containing time information extracted from event logs, are used to: (i) check time conformance; (ii) predict the remaining processing time of incomplete cases; (iii) recommend appropriate activities to end users working on these cases. In [15], an approach for predicting business process performances is presented. The approach is based on context-related execution scenarios discovered and modeled through state-aware performance predictors. In [24], the authors present a technique for predicting the delay between the expected and the actual arrival time of cases pertaining to a transport and logistics process. In [29], the queue theory is used in order to predict possible delays in process executions.

Another set of works in the literature focuses on approaches that generate predictions and recommendations to reduce risks. For example, in [8], the authors present a technique to support process participants in making risk-informed decisions with the aim of reducing the process risks. Risks are predicted by traversing decision trees generated from the logs of past process executions. In [25], the authors make predictions about time-related process risks by identifying and leveraging statistical indicators observable in event logs that highlight the possibility of transgressing deadlines. In [31], an approach for Root Cause Analysis (RCA) through classification algorithms is presented.

In the next section, the state of the art approaches with special process-log encoding are described.

3.2 Prediction with Complex Symbolic Encoding for structured and unstructured content

A next group of prediction approaches predict the outcome (e.g., the satisfaction of a business objective) of a case. Those can be clustered together as the methods that exploit the Complex Symbolic Sequences encodings of the process log.

Encodings for Complex Symbolic Sequences are always an adaptation of real data, that is in order to capture as much information as possible, and to have it in compact and ready for algorithmic processing form.

The paper [21] wonders about how it is possible to encode business log in a way that it will be used for clustering or for the subject of our interest, - predicting labels and next events.

They [21] give good all-around performer encoding for logs, called index-based encoding. They also give an alternative model called HMM-based encoding. HMM-based encoding is a direct extension to index-based, that takes into account the coefficient of the HMM model built for every possible outcome.

Evaluation is performed by building models that predict the label of the ongoing case.

Basically, they look at the log as in Table 1, and construct the feature vector of the form:

$$g_i = (s_i^1, ..., s_i^u, event_{i1}, ..., event_{im}, h_{i1}^1, ..., h_{i1}^r, ..., h_{im}^1, ..., h_{im}^r).$$

Where $s_i, i = \overline{1,u}$ are static features, $event_{ij}$ is an event class at each position, and h_{ik}^j the dynamic features associated.

Teinemaa extends this encoding with textual, unstructured content [34]. For this purpose they exploit methods from text mining, to extract features from textual attributes.

First step is to preprocess data. It is done in few steps. Firstly, text is segmented by tokenization. Then, the text is normalized, to eliminate word differences such as "e-mail" and "email". Later, *lemmatization* is used to bring same words of different grammatical form to the single form. Words such as "go" and "went" are grouped with the "go" label.

Next step is encoding. Four methods are proposed.

1. *Bag-of-n-grams* is a method, based on BoNG model. It has two parameters, n - maximum size of n-grams, and *idf* boolean variable specifying if the model is normalized. The n-gram means scoring feature representation based on the every sequence of n words. Normalization is used in order not to favor words that are relatively popular in one document.
   By the result of this procedure, the vector will be build in the following form:
   Having vocabulary:

$$V(1) = (\text{send}, \text{phone}, \text{radio}, \text{play}, \text{it}, \text{is}, \text{you}, \text{like}, \text{say}).$$

   And the input sentence "You said you like radio".
   The resulting vector will be of the form:
   $d^j = (0, 0, 1, 0, 0, 0, 2, 1, 1)$, assuming that parameters to the model is unigram, and no normalization is done.
   Downside of this model is that it can suffer from high dimensionality. Feature selection method is used to make this model more usable.
2. *Naive Bayes log count ratios* as an extension to the BoNG, adding weighting with NB log count ratios.
3. *Latent Dirichlet Allocation topic modeling* is a method, that represents text as a measure of a correspondence to some finite number of topics.
4. *Paragraph vector* is a technique that allows to represent the text as a feature vector of its meaning.

So, after one of these methods of text encoding is chosen, the corresponding vector is appended to the index-based encoding.

These encodings are used with classification methods such as support vector machines, decision trees.

3.3 Deep learning models

Recently, deep learning methods became exceptionally popular in research and industry communities. Business Process Management is not an exception[37,33,14]. Recent papers suggest state of the art performance on the prediction tasks for business process log outcomes.

Due to the nature of the problem, that is the sequence to sequence prediction, the recurrent neural networks are most exploited. The main motivation for this type of neural networks is currently the problems in Natural Language Processing, such as speech recognition[17], or translation[32].

The paper by Ilya[37], suggests the use of Recurrent Neural Networks with sliding window fashion. Which means that they use N events to predict the next one (to predict event k+1, they use k..(N+k) events).

The paper by Evermann[14] proposes the RNN network with LSTM cells with two hidden layers, 500 dimensions and 20 steps. They use batches of 20 to train the RNN with back propagation. The evaluation was made on the BPI2012, BPI2013 data sets. They argue that the results with this approach are comparable to state of the art based on clustering and annotating transition system approaches.

The most recent paper by Niek [33] uses RNN with LSTM cells to predict the next events. They achieve state of the art performance on most logs evaluated.

In order to predict next event and time, they use one-hot encoding for event sequence, and few time features (such as difference between time-stamps, time from midnight, time from beginning of the week). Using these features they construct a vector to be fed into recurrent neural network.

They are looking for functions f_a^1 and f_t^1 that are basically the probability distributions over all possible trace continuations.

Also, as they now have basically two sequences to train with, that are the sequence of events, and the sequence of time values, the paper suggests the different possible RNN architectures (As shown on figure 2).

The evaluation was done on many data sets, such as Helpdesk, BPI12, BPI12 with no duplicates, Environment permit. On most of them suffix (sequence of next events) prediction beats state of the art performance. It worked especially well with logs that had a lot of short traces (Helpdesk log contains average 7 event traces).

The problem investigated in this paper falls best into the problems discussed in this section i.e., into the set of very recent efforts aiming at predicting the sequence of future activities given the activities observed so far.

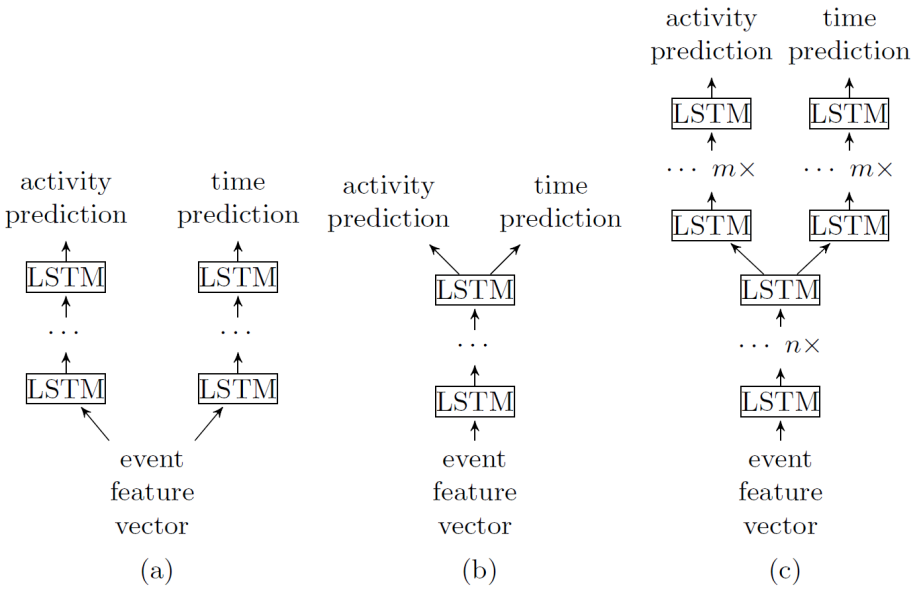

Fig. 2: Possible neural network architectures [33]. Single task layers (a), shared multitask layers (b), or n+m shared layers (c).

4 The Problem

Predictive business process monitoring methods use past process executions, stored in event logs, in order to build predictive that work at runtime to make predictions about the future. Among the different types of predictions about on an ongoing case, such as the remaining time or the fulfilment of a predicate, we can find the prediction of the sequence of future activities. This type of predictions can be useful in the scenario where some planning and resource allocation are needed for the running case. For instance, the hospital management can be highly interested in predicting the future activities of patients to be able to best organize machines and resources of the hospital.

Nonetheless, predicting sequences of activities is a quite complex and challenging task, as the longer the sequence is, the more difficult is to predict the most far-away activities. While predicting the sequence of future activities entirely from past execution data may be difficult, in real world scenarios, we often observe that some a-priori knowledge about the future of the running process executions exists

and could hence be leveraged to support the predictive methods and improve their accuracy.

For instance, in the hospital example, new medical guidelines may provide new knowledge on the fact that two treatments are not useful if used together in order to cure a certain disease, or that a certain screening is required in order to perform a specific surgery, or also that if a patient is allergic to a specific treatment she will never go to take it.

Also the exemplary knowledge known a-priory might be the weather information for the agricultural company that does the harvest. Knowledge about the weather can help a company to project on how their real processes concerning harvesting will unfold. For example assuming the weather is rainy in the middle of harvesting season, the big machinery usually used will not be able to perform needed operations. So inferring the incapability to use them will help to predict the real unfold of the processes. Thats could helps the management to guide their choices to best account for the new circumstances.

Moreover solved problem of this kind can bring much more benefits to the companies. That could be if the formalization of a problem be twisted. For example a big insurance company, that deals with thousands claims per day would decide that they need some changes such as cutting down jobs, they would have a way of empirically inferring the consequences. That means in addition to the classical qualitative methods of business process management that after building BPMN models have those changes analyzed my means of queuing theory or simulation one would have one more instrument in the toolkit. This algorithm would enable a process owner to make put some proposed changes as a-priori knowledge, and after predicting the continuation of selected business processes analyze it.

Given the practical motivations for the problem at hand let's dive once more into the research question identified.

RQ: How can prior knowledge about ongoing process be used to boost accuracy of predictions of trends of activities?

Here we decompose the research question into chunks to better understand the problem stated. *Boost accuracy of prediction* means to develop an approach that will have better performance than current state of the art. *Trends of activities* means that we are interested

in finding the suffix of the trace. *Ongoing process* means that we will have *prior knowledge* on the inference time.

This a-priori knowledge can be expressed in terms of LTL rules. For instance, in the hospital example, LTL can be used for defining the following rules:

1) `treatmentA` and `treatmentB` cannot be both used within the same course of cure of a patient:

$$\neg(\Diamond \texttt{treatmentA} \wedge \Diamond \texttt{treatmentB}) \tag{5}$$

2) `screeningC` is a pre-requisite to perform `surgeryD`:

$$(\neg \texttt{surgeryD} \sqcup \texttt{screeningC}) \vee \Box(\neg \texttt{surgeryD}) \tag{6}$$

3) `treatmentB` cannot be performed on this course of cure (e.g., because the patient is allergic to it):

$$\neg \Diamond \texttt{treatmentB} \tag{7}$$

In this paper, we aim at understanding whether and how a-priori knowledge can be leveraged in order to improve the accuracy of the prediction of the (sequence of the) next activity(ies) of an ongoing case in a reasonable amount of time. For instance, in the example of the hospital, being aware of the fact that `treatmentA` and `treatmentB` can never be executed together could help in ruling out a prediction of `treatmentB` whenever we have already observed `treatmentA` and vice versa.

Formally, given a prefix $p_k(\sigma) = \langle a_1, ..., a_k \rangle$ of length k of a trace $\sigma = \langle a_1, ..., a_n \rangle$ and some knowledge $\mathcal{K}(\sigma)$ on σ, the problem we want to face is to identify the function f such that $f(p_k(\sigma), \mathcal{K}(\sigma)) = s_k(\sigma)$.

5 The Solution

Predicting the suffix of a given prefix is a problem that is tackled by state-of-the-art approaches that make use of LSTM-based RNNs [14,33]. We hence start from these approaches and build on top of them to take into account a-priori knowledge.

Before presenting our approach, we need to observe that a basic solution that can be used to leverage a-priori knowledge for making predictions is the one provided by the inclusion of the a-priori knowledge in the data used for training the prediction model. However, this solution would raise a main practical problem: since the a-priori knowledge can in principle change from case to case, this would require to retrain the model for each prediction, thus hampering the scalability of the predictive system. A smarter approach is hence required for taking into account a-priori knowledge when predicting the future path of an ongoing case.

In the next sections, we first introduce an enhancement, called NOCYCLE, of state-of-the-art approaches for overcoming the issues encountered with traces characterized by a high number of cycles (Section 5.1). We then describe a further extension that allows us to take into account a-priori knowledge expressed in terms of LTL rules (Section 5.2). The extension algorithm for accounting for a-priori knowledge and the enhancement for dealing with cycles are then combined into the A-PRIORI technique.

5.1 Learning from Trace Structures

By experimenting the LSTM approach on different event logs, we found that event logs with traces containing a high number of repetitions of cycles perform worse than others, as also observed in [33]. This is mainly due to the fact that frequent repetitions of a cycle cause an increase in the probability distribution of the back-loop, i.e., the connection between the last and the first element of the cycle. To overcome this problem, we propose to equip Algorithm 1 with an additional function in charge of weakening such a back-loop probability. This function is composed of two parts: in the first part, the current trace is analyzed in order to discover possible cycles; in the second part, the cycle discovery is used for preventing the prediction of further repetitions of the cycle. More in detail:

1. For each prefix $p_k(\sigma) = \langle a_1 a_2 \dots a_k \rangle$ of size k, the algorithm checks if there are j ($j >= 2$) consecutive occurrences of a cycle $c = \langle a_{c_1} \dots a_{c_s} \rangle$, such that the last activity of the prefix corresponds to the last activity of the cycle $idx(a_k) = idx(a_{c_s})$;
2. j is then used to correct the distribution over different possible activities that can occur in the next position by decreasing the

probability of the first activity of the cycle a_{c_1} to occur again. To decrease this probability, the algorithm uses a coefficient, function of the number of cycle repetitions j, as a weight to adjust the probability distribution. Examples of formulas that can be used for this purpose are j^2 or e^j.

Algorithm 4 reports the pseudo-code of the NOCYCLE technique. Similarly to Algorithm 1 presented in Section 2.9, it takes as input a prefix $p_k(\sigma)$, the trained LSTM model $lstm$, and the maximum number max of iterations allowed. Then, it returns as output the complete trace (the prefix and the predicted suffix). In particular, the algorithm adds to the state-of-the-art Algorithm 1 the WEAKENPROB procedure described above to find cycles in the trace and decrease the probability of the first activity of the cycle to occur again at the end of a repetition. The resulting vector of weakened probabilities is hence used for getting the next symbol as in the basic procedure.

Algorithm 4 NOCYCLE extension for predicting the suffix of $p_k(\sigma)$

```
1: function PredictSuffixNoCycle(p_k(σ), lstm, max)
2:     k = 0
3:     trace = p_k(σ)
4:     do
5:         trace_encoded = encode(trace)
6:         next_symbol_prob = predictNextSymbols(lstm, trace_encoded)
7:         weak_next_symbol_prob = weakenProb (trace, next_symbol_prob)
8:         next_symbol = getSymbol(weak_next_symbol_prob, trace_encoded)
9:         trace = trace · next_symbol
10:        k = k + 1
11:    while (not next_symbol == end_symbol) and (k < max)
12:    return trace
13: end function
```

5.2 Learning from A-priori Knowledge

The overall idea for leveraging a-priori knowledge for predictive monitoring is simple: (i) we use the LSTM approach to get the possible predictions for an ongoing trace; that one can do, because the step by step output of a recurrent neural network is basically probability

distribution over next events in the trace. (ii) we rank them according to the likelihood of the prediction; and (iii) we select the first prediction which is compliant with the LTL rules describing the a-priori knowledge. However, although RNN inference algorithms are not computationally expensive per se, building all the possible predicted suffixes could be costly and inefficient. It is the standard problem of algorithms that rely on storing data in tree structures, where the growth of a tree is exponential. That is exactly the case here, as nevertheless of how many continuation of the trace is chosen at each step (let's day N steps), each next step of predictions will produce N^i nodes. Even for few iterations it is computationally and memory-wise not feasible to use this strategy.

Therefore, the alternative investigated in this paper leverage, on top of state-of-the-art LSTM techniques, the approach classically used in statistical sequence-to-sequence predictions in translation tasks [35], i.e., the *beamSearch* algorithm. The beamSearch is a heuristic algorithm based on partial ordering the probability trees, that is by exploring the search space only the most promising branches. In particular, we use the RNN architecture with LSTM cells and training system proposed in [33]. Then, in the testing phase, to predict a certain suffix, we use a new inference algorithm (A-PRIORI), which explores the probability space using beamSearch to cut the branches of the LSTM model which bring to predictions that are not compliant with the a-priori knowledge.

Algorithm 5 reports the pseudo-code describing the A-PRIORI algorithm. It takes as input the prefix $p_k(\sigma)$, the available a-priori knowledge $\mathcal{K}(\sigma)$, and the trained LSTM model $lstm$, together with three parameters: (i) $bSize$, which is the number of possible next symbols returned by the beamSearch algorithm; (ii) $maxSize$, which is the maximum number of branches that can be explored by A-PRIORI at the same time; and (iii) max, which is the maximum number of allowed iterations.

Intuitively, the algorithm iterates over a priority queue of prefixes, which is initialized with the input prefix $p_k(\sigma)$ (line 3) and that is used for regulating the number of branches to be explored. For each prefix in $prefixes$, $bSize$ ($< maxSize$) possible next activities are predicted (by means of the PREDICTNEXTSYMBOLS procedure of Algorithm 1 and the beamSearch algorithm) and for each prefix

Algorithm 5 A-PRIORI algorithm for predicting the suffix of $p_k(\sigma)$

```
1: function A-PRIORI(p_k(σ), K(σ), lstm, bSize, maxSize, max)
2:     h = 0
3:     prefixes = {p_k(σ)}
4:     while k ≤ max and not EMPTY(prefixes) do
5:         candidates_next = PREDICTPREFNEXTSYMBOLS(lstm, prefixes, bSize)
6:         top_candidates = TOPRANK(candidates_next, maxSize)
7:         for all candidate in top_candidates do
8:             if LAST_SYMBOL(candidate) <> end_symbol then
9:                 PUSH(candidate, prefixes)
10:            else
11:                if CHECK(candidate, K) then
12:                    return candidate
13:                end if
14:            end if
15:        end for
16:        h = h + 1
17:    end while
18: end function
```

$bSize$ new traces are obtained by concatenating the prefix with the corresponding $bSize$ predicted next activities (line 5). In this way, the algorithm generates $|prefixes| * bSize$ traces. In order to limit the search space, the algorithm ranks the predicted traces based on their estimated probability[1] and takes only the top $maxSize$ ones (line 6). For each of these traces (line 7), if the last symbol predicted is not the end symbol, the trace is added to $prefixes$ (line 9). Otherwise, if the trace is complete, the algorithm checks if it is compliant to the LTL rules in $\mathcal{K}(\sigma)$ (line 11). In this case, the trace is returned (line 12). The algorithm is then iterated until the queue of prefixes is empty or the maximum number of iterations max is reached (line 4).

5.3 Implementation

Algorithms 4 and 5 have been implemented in Python 2.6. In particular, the Keras [7] and TensorFlow [5] libraries have been used for

[1] Note that, in order to prevent overflow in the computation, the estimated probability for sequences of activities is computed as the sum of the logarithm of the probabilities of the next activities rather than as the product of the probabilities of the next activities.

Log	#Tr.	#Act.	avg-TL	avg-CR	Spars.
EnvLog	937	299	41	0.14	0.3191
HelpDesk	3804	9	3.6	0.22	0.0024
BPIC11	1 143	355	54	5.05	0.3897
BPIC12	13 087	7	7.5	1.35	0.0007
BPIC13	7 554	13	7	1.45	0.0017
BPIC17	31 509	27	18	0.46	0.0009

Table 2: The event logs

neural networks. The LTL[2] checker[2])for checking the compliance of traces to LTL rules is instead based on automata and written in Java.

As the problem of compatibility arose having Java and Python parts, the Py4J library[9] was used as a gateway to access Java code from Python.

The full source code is available on github[3]. The link given contain detailed readme with instruction on how to reproduce all experiments on any machine.

6 Evaluation

In this section, we provide an evaluation of our predictive business process monitoring techniques based on a-priori knowledge. In particular, we check: (i) whether the NOCYCLE algorithm leveraging knowledge about the structure of the process execution traces (and in particular about the presence of cycles) actually improves the predictions; and (ii) whether the A-PRIORI algorithm is able to leverage a-priori knowledge to improve the performance of the LSTM model.

6.1 Event Logs

For the evaluation of the techniques, we used six real-life event logs. Four of them were provided for the BPI Challenges (BPIC) 2011 [1], 2012 [12], 2013 [30], and 2017 [13], respectively. We also used two additional event logs, one pertaining to an environmental permit application process ("WABO"), used in the context of the CoSeLoG

[2] The LTL formula checker is taken from the open source code of ProM process mining suite.

[3] `https://github.com/yesanton/Process-Sequence-Prediction-with-A-priori-knowledge`

project [6] (*EnvLog* for short in this paper), and another containing cases from a ticketing management process of the help desk of an Italian software company (*Helpdesk*[4] for short).

1. **Environment permit log** (*EnvLog*) is the log containing data that come from Dutch municipality[5]. The cases are the application process of the environment permits. It contains 937 cases, 38.944 events, and 381 events types.
2. **Helpdesk** is the dataset containing events from a ticketing management process of the help desk of an Italian software company. The process consists of 9 activities, and all cases start with the insertion of a new ticket into the ticketing management system. Each case ends when the issue is resolved and the ticket is closed. This log contains 3804 process instances (a.k.a "cases") and 13710 events
3. **BPI 11 log** is the famous in process mining log taken from Dutch Academic Hospital. The cases of the log correspond to the Gynaecology department of the hospital. Phases consist of the diagnosis and treatment of the patient. Some attributes are repeating as the procedures could take place few number of times. As the log contains sensitive information it was anonymized.
4. **BPI 12 subprocess W dataset**[6] This is the part of the BPI 2012 challenge, that contains a real-life log, taken from a Dutch Financial Institute. The process represented in the event log is an application process for a personal loan or overdraft within a global financing organization.

   This log is contains a lot of information the usual prediction task system would bot be interested in. The paper[33] proposed to cut the log down to the traces what contain events that are manualy executed. So the pre-processed log was taken from their github repository[7].
5. **BPI 13 log**[8] was prepared by Volvo IT in Belgium. Log consist of incident and problem management from their system "VINST".

[4] `https://data.mendeley.com/datasets/39bp3vv62t/1`
[5] `https://data.4tu.nl/repository/uuid:e8c3a53d-5301-4afb-9bcd-38e74171ca32`
[6] `http://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f.`
[7] `https://github.com/verenich/ProcessSequencePrediction/tree/master/data`
[8] `http://www.win.tue.nl/bpi/doku.php?id=2013:challenge`

6. **BPI 17 log**[9] is a process taken from Dutch financial institute. In particular the process we used it a loan applications. It has much more data that other logs that we considered.

The characteristics of these logs are summarized in Table 2. For each log, we report the total number of traces, the number of activity labels (i.e., the size of the activity set of the log), the average trace length (avg-TL), the average number of repetitions of all cycles in the log (avg-CR), and the ratio between the number of activity labels and the number of traces, indicating the sparsity of the activity labels over the log.

6.2 Experimental Procedure

In order to evaluate the techniques presented in the paper, we adopted the following procedure. For each event log:

1. We divided the event log in two parts: a **training set** composed of the 67% of the whole event log and a **testing set**, composed of the remaining 33%.
2. We derived the a-priori knowledge on the traces of the testing set as follows. We randomly selected 10% of traces of the testing set. From each trace, we extracted 4 prefixes of lengths corresponding to the 4 integers in the interval $[mid - 2, mid + 2]$, where mid is half of the median of the trace lengths. We derived all suffixes of these prefixes and we used the DeclareMiner ProM plug-in [22] to discover LTL rules satisfied in all those suffixes (Figure 3 shows the LTL extraction from DeclareMiner). Then, we defined 2 conjunctive rules describing a *strong a-priori knowledge* and a *weak a-priori knowledge*, which respectively strongly and weakly constrain the traces. In particular, we discovered rules of type $\Diamond A$ (which imposes the occurrence of A) for defining the weak a-priori knowledge and rules of type $\Box(A \rightarrow \Diamond B) \wedge \Diamond A$ (which imposes the occurrence of both A and B and that every occurrence of A is followed by an occurrence of B) for defining the strong a-priori knowledge. For the weak a-priori knowledge, we randomly selected from the discovered rules one, two or three (depending on the average length of the traces in the log) rules of type $\Diamond A$ and

[9] `http://data.4tu.nl/repository/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b`

Log	A-priori Strong	A-priori Weak
EnvLog	$\Box(a \rightarrow \Diamond b) \wedge \Diamond a \wedge \Box(c \rightarrow \Diamond d) \wedge \Diamond c$	$\Diamond a \wedge \Diamond c$
HelpDesk	$\Box(e \rightarrow \Diamond f) \wedge \Diamond e$	$\Diamond e$
BPIC11	$\Box(g \rightarrow \Diamond h) \wedge \Diamond g \wedge \Box(i \rightarrow \Diamond l) \wedge \Diamond i \wedge \Box(m \rightarrow \Diamond n) \wedge \Diamond m$	$\Diamond i \wedge \Diamond h \wedge \Diamond o$
BPIC12	$\Box(p \rightarrow \Diamond q) \wedge \Diamond p$	$\Diamond p$
BPIC13	$\Box(r \rightarrow \Diamond s) \wedge \Diamond r \wedge \Box(t \rightarrow \Diamond r) \wedge \Diamond t$	$\Diamond s \wedge \Diamond r$
BPIC17	$\Box(u \rightarrow \Diamond v) \wedge \Diamond u$	$\Diamond u$

Table 3: The a-priori knowledge.

we composed them into a single conjunctive formula. Similarly, for the strong a-priori knowledge, we randomly selected one, two or three rules from the discovered rules of type $\Box(A \rightarrow \Diamond B) \wedge \Diamond A$ and we composed them into a single conjunctive formula. We followed this systematic procedure for defining the a-priori knowledge, to limit the bias of the selected rules while guaranteeing that they are satisfied in a reasonable number of traces of the testing set. The schematic form of the rules used in the evaluation is reported in Table 3, where - for the sake of readability - we replace the original activity names with single characters. Starting from strong and weak a-priori knowledge, we built a *strong a-priori testing set* and a *weak a-priori testing set*, respectively composed of the subsets of traces of the testing set that satisfy strong and weak a-priori knowledge.

3. we compared NOCYCLE and A-PRIORI[10] against a baseline provided by the technique presented in [33]. For each technique, we computed: (i) the length of the predicted suffixes; and (ii) their similarity with the prediction ground truth measured using the Damerau-Levenshtein similarity [10].

The experiments have been performed both on a GPU Tesla K40c and on a conventional laptop with Code i5 CPU. As for the LSTM training settings we used the ones identified by Tax et al. [33] as the most performing ones for facing the problem of predicting sequences of future activities.[11] The time required for training the LSTM neural

[10] We set *bSize* to 3 and, for the coefficient in charge of weakening the probabilities of activities in a cycle, we used the exponential formula (e^j, where j is the number of cycle repetitions).

[11] We used an architecture characterized by two LSTM layers. The algorithm used is the Adam learning algorithm with categorical cross entropy loss and the dropout coefficient has been set to 0.2.

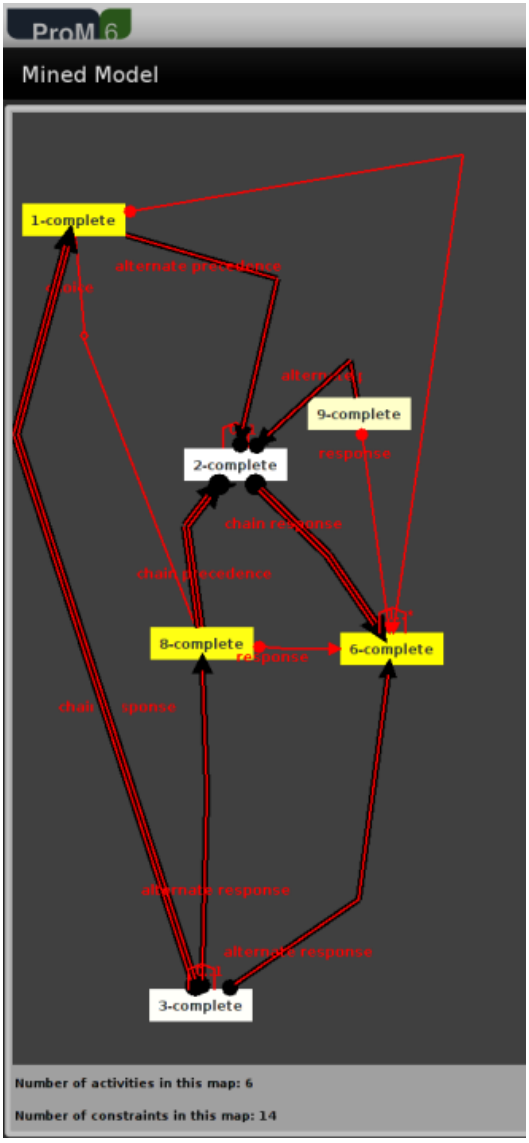

Fig. 3: Declare Miner in action

network is about 2 minutes per epoch using the GPU and 15 minutes using the CPU. The inference time for NOCYCLE is about 0.1-2 seconds per trace (depending on the log), whereas the inference time for A-PRIORI is 4 times higher on average.

Log	Baseline	Nocycle	A-priori	Groundtruth	Tested	Prefix
EnvLog	0.250 [17.4]	0.250 [17.4]	0.070 [95.00]	29.40	80	19 – 22
HelpDesk	0.551 [1.44]	0.551 [1.44]	0.831 [2.36]	3.00	576	2 – 5
BPIC11	0.204 [199.00]	0.281 [199.00]	0.274 [198.00]	117.11	144	13 – 16
BPIC12	0.071 [47.07]	0.387 [6.86]	0.416 [7.53]	10.95	1 548	2 – 5
BPIC13	0.116 [100.80]	0.502 [14.71]	0.596 [6.13]	7.15	3 209	2 – 5
BPIC17	0.448 [11.78]	0.448 [11.78]	0.510 [15.14]	16.01	10 153	6 – 9

Table 4: Prediction results on the strong a-priori testing set

Log	Baseline	Nocycle	A-priori	Groundtruth	Tested	Prefix
EnvLog	0.246 [18.22]	0.246 [18.22]	0.084 [89.91]	31.31	108	19 – 22
HelpDesk	0.551 [1.44]	0.551 [1.44]	0.747 [2.14]	3.00	576	2 – 5
BPIC11	0.220 [199.00]	0.292 [199.00]	0.282 [197.4]	112.66	450	13 – 16
BPIC12	0.100 [48.08]	0.263 [6.81]	0.264 [7.00]	8.33	3 179	2 – 5
BPIC13	0.130 [95.19]	0.459 [14.92]	0.569 [5.14]	5.85	4 364	2 – 5
BPIC17	0.448 [11.78]	0.448 [11.78]	0.469 [14.00]	16.01	10 153	6 – 9

Table 5: Prediction results on the weak a-priori testing set

6.3 Results and Discussion

Tables 4 and 5 report, for each event log, the performances of the two techniques we propose on the strong a-priori and weak a-priori testing sets. Also, one can take a look at the figure 4 where results are visualized. The results for both testing sets are compared with the baseline presented in [33]. For each log, we provide the average Damerau-Levenshtein similarity between the predicted sequence (in square brackets, its average length) and the ground truth (column 5). The best average Damerau-Levenshtein similarity for each log is emphasized in gray. Column 6 reports the number of traces tested while column 7 specifies the range of the prefix lengths used for the specific event log.

The tables show that the proposed algorithms outperform the baseline in most of the logs. The presence of cycles in the logs has a strong impact on the performance of the Nocycle algorithm. In particular, if the logs have an average number of cycle repetitions smaller than 0.5, as in the case of EnvLog, HelpDesk and BPIC17, then Nocycle does not show any improvement over the baseline. Therefore, we can conclude that Nocycle correctly deals with the presence of cycles in the logs to improve the predictions.

A-priori perform worse on logs EnvLog and the BPIC11. The reason for this can be explained by the fact that, in these two logs, ac-

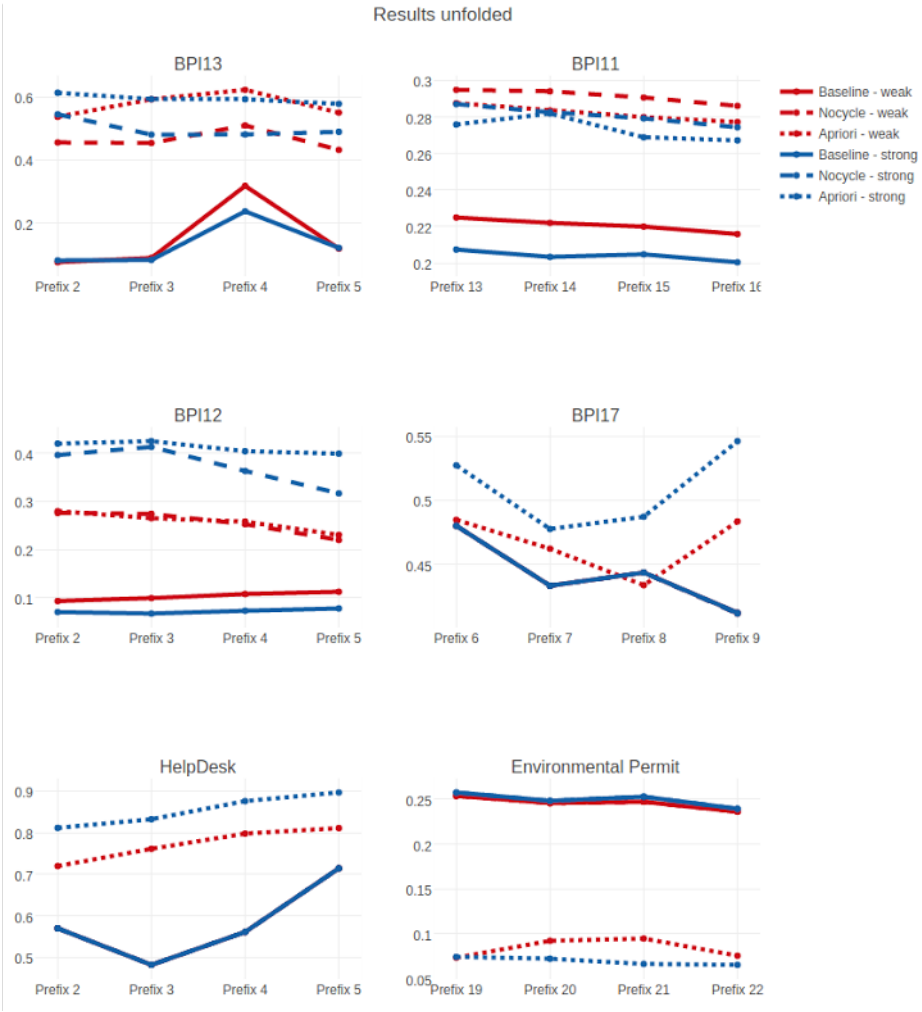

Fig. 4: Graphical representation of evaluation

tivity labels are sparse with an unusually high number of labels with respect to the number of traces. Indeed, Table 2 shows that the ratio between the number of activity labels and the number of traces (column 8) for these logs is higher with respect to the other logs. We can also notice that the availability of highly constraining rules in the a-priori knowledge improves the performance of A-PRIORI. Therefore, we can conclude that A-PRIORI is able to correctly leverage a-priori knowledge in a way that it performs better when the activity set

of the log is not particularly large (and the log does not contain sparse behaviors) and when the a-priori knowledge constrains more the process behavior.

7 Conclusions

In the thesis, we have presented two techniques based on RNNs with LSTM cells able to leverage knowledge about the structure of the process execution traces as well as a-priori knowledge about their future development for predicting the sequence of future activities of an ongoing case. In particular, we show that opportunely tailoring LSTM-based algorithms it is possible to take into account a-priori knowledge at prediction time without the need to retrain the predictive algorithms in case new knowledge becomes available. The results of our experiments show that NOCYCLE correctly deals with the presence of cycles in the logs and A-PRIORI is able to correctly leverage a-priori knowledge in a way that it performs better with logs characterized by a low degree of sparsity of activity labels and when the a-priori knowledge constrains the behavior of the process more.

Future work will include: (i) dealing with more complex forms of a-priori knowledge. In particular, we aim at addressing a-priori knowledge on activities and on their data payload, as well as dynamic knowledge that can evolve in the future of an ongoing case; that might include other ways of representing the knowledge. It might be possible to extend LTL rules with some case specific semantics; (ii) extending the proposed algorithms to leverage a-priori knowledge also for other types of predictions; (iii) extending the experimental evaluation especially focusing on the investigation of metrics for evaluating the influence on the predictions of the different degrees of freedom/strength of the a-priori knowledge; and (iv) inserting the presented techniques in predictive business process monitoring frameworks such as the one discussed in [11].

References

1. 3TU Data Center: BPI Challenge 2011 Event Log (2011), doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54
2. van der Aalst, W.M.P., de Beer, H.T., van Dongen, B.F.: Process Mining and Verification of Properties: An Approach Based on Temporal Logic, pp. 130–147. Springer Berlin Heidelberg, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/11575771_11
3. van der Aalst, W.M.P., Pesic, M., Song, M.: Beyond Process Mining: From the Past to Present and Future, pp. 38–52. Springer Berlin Heidelberg, Berlin, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-13094-6_5
4. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Information Systems 36(2), 450–475 (2011)
5. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), \url{http://tensorflow.org/}, software available from tensorflow.org
6. Buijs, J.: Environmental permit application process ("wabo"), coselog project - municipality 4 (2014), https://doi.org/10.4121/uuid:e8c3a53d-5301-4afb-9bcd-38e74171ca32
7. Chollet, F.: Keras. https://github.com/fchollet/keras (2015)
8. Conforti, R., de Leoni, M., Rosa, M.L., van der Aalst, W.M.P.: Supporting risk-informed decisions during business process execution. In: Proc. of CAiSE 2013. pp. 116–132. Springer (2013)
9. Dagenais, B.: py4j: A bridge between python and java (2016), \url{https://www.py4j.org}, software available from py4j.org
10. Damerau, F.J.: A technique for computer detection and correction of spelling errors. Commun. ACM 7(3), 171–176 (Mar 1964)
11. Di Francescomarino, C., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Transactions on Services Computing PP(99) (2016)
12. van Dongen, B.: Bpi challenge 2012 (2012), http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
13. van Dongen, B.: Bpi challenge 2017 (2017), https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
14. Evermann, N., Rehse, J.R., Fettke, P.: A deep learning approach for predicting process behaviour at runtime. In: PRAISE-2016 (2016)
15. Folino, F., Guarascio, M., Pontieri, L.: Discovering context-aware models for predicting business process performances. In: Proc. of On the Move to Meaningful Internet Systems (OTM), pp. 287–304. Springer (2012)
16. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning, chap. Sequence Modeling: Recurrent and Recursive Nets, pp. 373–420. MIT Press (2016)
17. Graves, A., Mohamed, A.R., Hinton, G.E.: Speech recognition with deep recurrent neural networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 6645–6649. IEEE (2013)
18. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)

19. Kang, B., Jung, J.Y., Cho, N.W., Kang, S.H.: Real time business process monitoring using formal concept analysis. Industrial Management & Data Systems 111(5), 652–674 (2011), http://dx.doi.org/10.1108/02635571111137241
20. Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., Herbst, E.: Moses: Open source toolkit for statistical machine translation. In: Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions. pp. 177–180. ACL '07, Association for Computational Linguistics, Stroudsburg, PA, USA (2007), http://dl.acm.org/citation.cfm?id=1557769.1557821
21. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: BPM 2015, pp. 297–313 (2015)
22. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient Discovery of Understandable Declarative Process Models from Event Logs, pp. 270–285. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-31095-9_18
23. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: CAiSE 2014. LNCS, vol. 8484, pp. 457–472. Springer (2014)
24. Metzger, A., Franklin, R., Engel, Y.: Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In: Proceedings of the 2012 Annual SRII Global Conference. pp. 313–322. SRII '12, IEEE Computer Society, Washington, DC, USA (2012)
25. Pika, A., van der Aalst, W.M.P., Fidge, C.J., ter Hofstede, A.H.M., Wynn, M.T.: Predicting Deadline Transgressions Using Event Logs, pp. 211–216. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
26. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977)
27. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Time and activity sequence prediction of business process instances. CoRR abs/1602.07566 (2016)
28. Schonenberg, H., Weber, B., van Dongen, B., van der Aalst, W.: Supporting Flexible Processes through Recommendations Based on History, pp. 51–66. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-85758-7_7
29. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining for delay prediction in multi-class service processes. Inf. Syst. 53, 278–295 (2015)
30. Steeman, W.: Bpi challenge 2013 (2013), https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07
31. Suriadi, S., Ouyang, C., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Root Cause Analysis with Enriched Process Logs, pp. 174–186. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
32. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Proceedings of the 27th International Conference on Neural Information Processing Systems. pp. 3104–3112. NIPS'14, MIT Press, Cambridge, MA, USA (2014), http://dl.acm.org/citation.cfm?id=2969033.2969173
33. Tax, N., Verenich, I., Rosa, M.L., Dumas, M.: Predictive business process monitoring with lstm neural networks (2017), to appear in CAiSE 2017
34. Teinemaa, I., Dumas, M., Maggi, F.M., Di Francescomarino, C.: Predictive business process monitoring with structured and unstructured data. In: BPM 2016. pp. 401–417 (2016)

35. Tillmann, C., Ney, H.: Word reordering and a dynamic programming beam search algorithm for statistical machine translation. Comput. Linguist. 29(1), 97–133 (Mar 2003)
36. VAN DER AALST, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems and Computers 08(01), 21–66 (1998), http://www.worldscientific.com/doi/abs/10.1142/S0218126698000043
37. Verenich, I., Dumas, M., Rosa, M.L., Maggi, F.M., Chasovskyi, D., Rozumnyi, A.: Tell me what's ahead? predicting remaining activity sequences of business process instances (June 2016), http://eprints.qut.edu.au/96732/
38. Xing, Z., Pei, J., Keogh, E.: A brief survey on sequence classification. SIGKDD Explor. Newsl. 12(1), 40–48 (Nov 2010), http://doi.acm.org/10.1145/1882471.1882478