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
Supervisor: Chiara Ghidini

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 this 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

Nowadays, the success of a company is highly correlated with its ability for businesses to provide superior delivery of services compared to its rivals. Competition makes companies seek for better management and organization of their work-flows. Furthermore, technologies have also evolved, enabling the use of advanced methods for analyses that were not possible before. To assist business needs, Process Mining field has evolved. It designs techniques to analyze a performance of a company from event logs.

Predictive monitoring [25] 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, a degree of compliance with respect to a measure, or achieving an objective.

An example would be predicting the compliance of a process execution in an insurance company, given the record of past processes and their compliance outcomes. 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 on predicting an outcome (class label of the process). That would be having some predefined set of classes that the traces in the log can belong to. For example, the classes could be "deadline met" and "deadline violated". As an example, an issue to resolution process can have defined maximum 7 days for the process to be considered successfully resolved. In this setup having the process just started, one can understand and predict one of these labels i.e. if the process will have a delay (a duration higher than 7 days) or not.

Also, methods that focus on predicting ongoing process execution are developed. That is predicting the next activities that will most likely be performed in the sequence after those that already happened. This thesis focuses on this type of predictions.

Given past executions of the process (event log consisting of traces) as a set of feature vectors in classic machine learning, our aim would be to predict future paths in a process.

However, even if many sophisticated methods [14,29,36] are currently available for prediction of future paths none of them account for the prior knowledge that can be additionally available on execution time.

Indeed, 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 an 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 with the following 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 [19] 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 [36] 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 thesis 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 large 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 processes 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).

Practically speaking, the log is a set of process executions (Traces). Trace is a set of events happened in sequence in a process execution.

The sequence is defined by timestamps. They are the first essential part of the process. Every correctly stored process must contain the time stamps of when certain activity was performed.

Trace should have an identification, that is the case ID. Whenever a real world process starts the new case ID is created. For example, in the issue to resolution process, the ID refers to a unique number associated with one instance of the issue raised.

The third essential element of a trace is an Activity label. These are the names of different activities performed during one instance of the process. For our example these might be "issue recorded", "decision made", "confirmation received", or "issue canceled". These provide an understanding of what the process is actually performing.

Many other data values can be recorded. A real world example is shown in the figure 1 from a Dutch hospital log. The figure shows one event of the process. This representation refers to the standard XES [40] to represent logs.

```xml
<event>
    <string key="org:group" value="General Lab Clinical Chemistry"/>
    <int key="Number of executions" value="1"/>
    <int key="Specialism code" value="86"/>
    <string key="concept:name" value="ordertarief"/>
    <string key="Producer code" value="CRLA"/>
    <string key="Section" value="Section 4"/>
    <int key="Activity code" value="379999"/>
    <date key="time:timestamp" value="2005-09-07T01:00:00.000+02:00"/>
    <string key="lifecycle:transition" value="complete"/>
</event>
```

Fig. 1: Trace example, exempt from the BPI11 hospital log [1]

One can see that the timestamp is present and in XES activity labels are represented through the keyword "concept:name".

There are a number of other parameters such as "org:group" that shows the department in the clinic, "Number of executions" that shows a number of times certain operation was performed and others. All parameters combined give the best picture of the process execution, therefore it is important for methods exploiting logs to use most complete data set possible.

Furthermore, 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 [37].

As for example in the hospital log mentioned above, it might be a parameter with comments from the doctor for patient wellbeing.

In the data based systems, the log can be stored in different structures. Enterprise Resource Planning (such as SAP, Oracle) systems have their own data format of journaling the data. It can also be CSV spreadsheet (figure 2) or XES formatted log (shown in figure 1).

2.2 Process mining

Process mining is a business process management technique that enables the analysis of processes based on the event logs. The main

case:concept:name	event:concept:name	Treatment code	Producer code	Diagnosis	End da
0	1e consult poliklinisch	103	SRTH	maligniteit cervix	
0	administratief tarief - eerste pol	103	SRTH	maligniteit cervix	
0	verlosk.-gynaec. korte kaart kosten-	103	SGEH	maligniteit cervix	
0	echografie - genitalia interna	103	SGEC	maligniteit cervix	
0	1e consult poliklinisch	103	SGEH	maligniteit cervix	
0	administratief tarief - eerste pol	103	SGEH	maligniteit cervix	
0	simulator - gebruik voor aanvang me	103	RATH	maligniteit cervix	
0	behandeltijd - eenheid t3 - megavolt	103	RATH	maligniteit cervix	
0	teletherapie - megavolt fotonen best	103	RATH	maligniteit cervix	
0	aanname laboratoriumonderzoek	103	CRLA	maligniteit cervix	

Fig. 2: Trace example (CSV format), exempt from the BPI11 hospital log [1]

idea is improving business processes based on analysis of the logs of sequential executions of business processes. This is what makes it different from the methods established in data mining and business intelligence which treat the data as a set of distinct events to gather insights.

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

- **Process discovery:** encompasses all methods to discover insights from the past process executions. As an example would be the issue to resolution process from the insurance company. Process discovery methods allow building abstract representation of an event log. The goal of this is to build approximated model that delivers better insights into the underlined processes. Discovered models can be in forms of BPMN models, Petri nets, and declarative rules.
- **Conformance analysis:** it takes existent formalized models and check them against real logs. In detail, that means that the business process analyst has the BPMN model for the process created (or generated). And now, the goal is to match the real processes to the model (mostly interest is in doing it on a fly). That is done in order to check for any deviations the process can have, and if possible for prevention of undesired behavior.
- **Process enhancement:** it helps to use logs in order to enhance existing process models. The information of the past is used to change the model based on how the processes are being performed.

2.3 Predictive process monitoring

Predictive monitoring is an advancing subfield of Process Mining. As a subfield of Process Mining, its main focus also lies in exploring and exploiting the process logs (the process executions). More specifically, predictive monitoring encompasses the methods that deal with online predictions of process outcomes. The outcome of interest can be compliance of with a measure (for example in an order to cash process the measure can be that any of the financial operations that involve an amount more than 2500 euro, are not performed without confirmation from the top financial officer), achieving objective (such as in issue to resolution process, if the process will be completed in under 7 days time), or classifications (giving a label to the uncompleted trace, such as issue resolved, or not).

In predictive monitoring historical traces are used to train predictive models, then it is used at runtime to predict the future behavior of a new, unseen ongoing trace. To do this, the information contained in the current trace prefix is used to query the predictive model and to get a prediction on the current trace.

Many algorithms are applied to solve the above-mentioned problems, for example, classic machine learning algorithms as classification and clustering [23]. More complex concepts are also used, such as Markov models [23], time series [22], and deep learning [36,14,41] approaches.

In order to successfully apply these algorithms, it is necessary to have a solid understanding of the process log nature. The aspects that can come into play, are the data types (while the case ID and activity labels are defined as categorical values, we have to deal with time stamps that are basically real-valued, also text comments that are strings or other numerical values such as amounts of money), stationarity (it usually refers to time series data that have a stable mean, variance etc. over time. The best example from process logs is timestamps. One should account that they represent only the moment in an absolute time scale, while algorithms would only work if the input will be a relative measure, such as a difference between timestamps).

Keeping in mind the definition of the business process log from section 2.1 one can easily find the caveats in the early process mining approaches, that tend to ignore the data payload, taking into con-

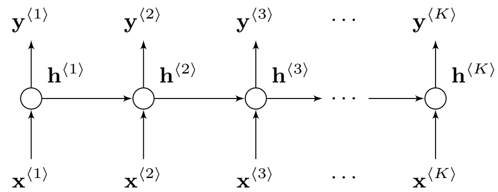

Fig. 3: Recurrent Neural Network

sideration only the control-flow (sequence of activities). Some of the approaches do the opposite, that is consider data, without control-flow [3,4,31]. In that case, the traces are considered to be simple sequences. Moreover, methods introduced are usually designed to work in offline fashion. But, to make more accurate, mature, online predictions one should leave out neither payload nor control-flow information.

2.4 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 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 learned during the training phase.

Recurrent Neural Networks (RNNs, see Fig. 3) are a subclass of Neural Networks. We illustrate them with the help of an example in which the classification task concerns the assignment of 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 [17]). As shown in Fig. 3, 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 learned during the training phase.

Among the different cell functions h (see equation (3)) explored in the literature, Long Short-Term Memory (LSTM) [19] 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 thesis.

2.5 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 [36] relies on an encoding of activity sequences that combine features related to the activities in the sequence (the so called *one-hot encoding*).

One-hot encoding is actually the $v * m$ sized matrix, where each row represents one step of a business process. One-hot is because the table can only contain the binary value of one or zero (even though some extended versions of one-hot encoding are used to add information of features such as time [36]).

Even though the table is usually large and not so efficient memory-wise, it is suitable for neural networks computations (as most of the operations are multiplications, it's easy to multiply and add ones and zeros).

If we turn the log from figure **??** into the one-hot encoding it will have representation shown in table 1 (we use only control-flow in this example, that is the activity label).

	A_SUBMITTED	A_PARTLYSUBMITTED	A_DECLINED
e_1	1	0	0
e_2	0	1	0
e_3	0	0	1

Table 1: Example of One-hot encoding

Given the set $A = \{a_{1_A}, \ldots a_{m_A}\}$ of all possible activities, an ordering function $idx : A \to \{1, \ldots, |A|\} \subseteq \mathbb{N}$ 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 $idx : A \to \{1, 2, 3\}$ 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 [36] 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).

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

1: **function** PREDICTSUFFIX($p_k(\sigma)$, $lstm$, max)
2: $\quad k = 0$
3: $\quad trace = p_k(\sigma)$
4: $\quad$ **do**
5: $\qquad trace_{encoded}$ = ENCODE($trace$)
6: $\qquad next\_symbol\_probs$ = PREDICTNEXTSYMBOLS($lstm$, $trace_{encoded}$)
7: $\qquad next\_symbol$ = GETSYMBOL($next\_symbol\_prob$, $trace_{encoded}$)
8: $\qquad trace = trace \cdot next\_symbol$
9: $\qquad k = k + 1$
10: $\quad$ **while** (not $next\_symbol$ == end_symbol) and (k <max)
11: $\quad$ **return** $trace$
12: **end function**

Testing set is used to evaluate the algorithm. Using prefixes of a chosen size, trained model is used to predict the continuations. Then predicted suffixes are aligned with real suffixes of the input traces using evaluation metrics.

As an evaluation metrics, predictive business monitoring adopted known in NLP community Damerau-Levenshtein similarity metric [10]. Damerau-Levenshtein similarity is defined as normalized Damerau-Levenshtein distance. The distance is computed by counting the minimum number of operations needed to change one trace into another. Those operations are an insertion of a missing event, deletion of an event, substitution of a single event, or transposition of two consecutive events.

2.6 Linear Temporal Logic

Linear Temporal Logic [28] (LTL) is a modal logic with modalities devoted to describing 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.

2.7 Beam search

Beam search is a heuristic search algorithm, based on the principle of exploring graph of possible solutions using the most promising node. It is built for dealing with large search spaces. Specifically, it orders all partial solutions with some heuristic measure; keeps only a small portion of the most promising solution and expands them to get the complete solution.

Beam search is used in areas such as machine translation[21].

The algorithm is used to find sequences with certain characteristics that are conceptualized by evaluation function.

There are few variants of this algorithm, two of which will be thoroughly explained here: The best-first and the breadth-first.

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 (open state means that the node of the tree was not visited yet, and can be explored in the next iteration, while closed means that the node was already visited). 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 described in the listing 2.

Algorithm 2 Best first search

```
 1: initialize the initial state as OPEN
 2: while OPEN set is not empty do
 3:     Take the best node from the list (call it N), and make it CLOSED
 4:     if N is a solution then
 5:         success, return the solution traced back
 6:     end if
 7:     Expand node N (find all successors)
 8:     for all node of the successors do
 9:         if node is not in CLOSED then
10:             apply evaluation function f to the node
11:             node as OPEN
12:         end if
13:     end for
14: end while
```

On the first line, the graph is initialized by adding initial state to the set of OPEN states. Second line starts the loop, that finishes when the solution found (according to evaluation function), or the tree was fully explored (no events are left in OPEN set). Next, on line 3, the best (according evaluation function f) element is chosen, and moved from OPEN set to CLOSED. On lines 4-6 algorithm checks if chosen element represent a solution. If not on the line 7 all successors of the element are found. The remaining of the pseudo code reports exploring those succcessors and adding them to the list of OPEN nodes.

Different evaluation functions can be used for the task. The decision is made upon the specifics of the problem, and it directly influences the effectiveness of an algorithm. The example of the evaluation function can be the count of events if we need the prediction to be short, or if the predicted sequence is conformant with a defined rule.

Breadth-first variation of Beam search The breadth-first search begins with an empty node and explores the search space in the breadth-wise order. This type of beam search operates using few priority queues instead of tree structures as in the previous approach.

The algorithm is reported in the pseudo code listing 3 (all elements in priority queues are stored with priority based on some evaluation function)

The algorithm starts with defining two priority queues on line 1. On line 2, the element given as an input is appended to first queue. The third line defines the condition that calculations will stop that is if the solution tree has reached its bottom (the priority queues s1 and s2 are empty). Line 4 starts the while loop that will iterate over all elements in s1. Lines 5-9 describe that each element from $s1$ either a solution (in that case line 7 returns it) or the next elements are predicted (based on the *elem*) and top *beamsize* elements are chosen to be added to the $s2$ queue. On line 11, the algorithm is reset for the next step of the beam search. Only best *beamsize* elements are chosen (others are discarded).

Algorithm 3 Breath-first Beam search

1: define two empty priority queues, $s1$ and $s2$
2: add the initial element to $s1$
3: **while** solution is not found **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:     initialize $s2$ with the *beamsize* best elements from $s1$
12: **end while**

Both the beam search algorithms use the same tactic: they cut the search space as to keep only the most probable and better performing prospective solutions. These algorithms are different, and depending on the nature of the problem they can have very different performance.

3 Related Work

In the following sections, the techniques and algorithms used in recent papers on predictive process monitoring that are considered as a state of the art to the problems of interest will be described. These approaches are also discussed in terms of their suitability to be used for predicting trends of activities using a-priori knowledge.

The methods are ordered in a chronological manner, each has a description of how it relates to the above-mentioned problem and a short discussion on the results and applications.

Firstly, we start off by giving a short overview of early approaches concerning predictive monitoring. Secondly, the state of the art methods based on Complex Symbolic Sequences Encodings will be described. Finally, the overview of Deep learning based 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 [39]. 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. In particular, the Petri nets were used for predictions of the business process duration [30].

Petri nets have few important properties such as having a formal semantics. Also, they are simple models and have graphical nature. Their properties are mathematically investigated.

Kang and Jung [20] propose a new approach for monitoring the progress of ongoing processes and predicting probable performances and routes. Their contribution is based on the use of 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 evolving (next activities) of ongoing processes by introducing concept lattice and reachability lattice. Also, to the first group of works, we can associate the work in [4]. In this work, the authors 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 [16], 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 [26], 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 [32], queue theory is used in order to predict possible delays in process executions.

Another group 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 process risks. Risks are predicted by traversing decision trees generated from logs of past process executions. In [27], 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 [34], an approach for Root Cause Analysis (RCA) through classification algorithms is presented. These methods would suffer from low accuracy if applied in accordance of the problem stated. They usually state the probability of the process to "derail", but not predicting the full sequence of events.

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

The second group of approaches focused on predicting the outcome (e.g., the satisfaction of a business objective) of a case can be clustered together as the methods that exploit Complex Symbolic Sequences encodings of the process log.

Complex Symbolic Sequences are introduced [23]. They allow to capture a lot information about a log. Complex Symbolic Sequence is basically ordered list of vectors, where each vector is a subset of some alphabet inferred from the log.

When Complex Symbolic Sequences are used instead of Simple Symbolic Sequences (encoding that focuses solely on activity labels) for dealing with prediction problems, the complexity space of the prediction problem grows substantially. This suggests using 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 [23].

To exemplify this approach for encodings let's again take a look at the log in figure ??. The Complex Symbolic Sequence form of this log will have following representation:

	$concept : n_1$	..	$concept : name_3$	$org : resource_1$	..	$org : resource_3$	$time : timestamp_1$	..
tr1	A_SUBMITTED	..	A_DECLINED	112	..	10929	2011-12-12T16:06:11	..
tr2		..	..	..	..	..	..	..

Table 2: Example of Complex Symbolic Sequence encoding

Complex Symbolic Sequences allow us to encode a real log, in order to capture as much information as possible while having a structure that easily can be used with standard machine learning algorithms.

In [23] the authors explore different types of encodings, called index-based encodings. 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 coefficients of the built HMM model for every possible outcome. In particular for binary classification, the log is split into two parts (according to the class) and two HMM models are built.

Basically, they look at the log, and construct a 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 activity label event class at each position, and h_{ik}^j are the dynamic features associated to each event.

In [37] the authors extends this encoding with textual, unstructured content. For this purpose they exploit methods from text mining, to extract features from textual attributes. These encodings are used with classification methods.

When Complex Symbolic Sequence encoding needs to have textual data encoded as additional parameters. A text knowledge extraction techniques, as described below are used.

First data is preprocessed is three steps: tokenized, normalized, and lemmatization is made.

Next step is encoding. The most used methods [37] are: *Bag-of-n-grams*, *Naive Bayes log count ratios*, *Latent Dirichlet Allocation topic modeling*, and *Paragraph vector*.

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

This representation also has its downsides, even though it captures much more information. It is impossible to represent the complete log in this way because one needs to limit the number of events in the traces to some definite number. In our example in the table 2 the trace length was chosen to be 3. This puts a limitation on the power of this encoding.

Before mentioned approaches focus on binary classification of the traces. The problem we are solving is different as instead of predicting a binary outcome of the first N events, we predict the next events. That makes the number of choices bigger (from 2 in binary classification to K as a number of different possible next events). The methods described in this section, if generalized to capture the prediction of next events, would become unusable.

3.3 Deep learning approaches

Recently, deep learning methods became exceptionally popular in research and industry communities. Business Process Management is not an exception [41,36,14]. Recent papers suggest state of the art performance on the prediction of next events and time features of the current trace.

Due to the nature of the problem, i.e. 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[18], or translation[35].

The paper by Verenich [41], suggests the use of Recurrent Neural Networks with sliding windows. Sliding windows is using N events to predict the next one (to predict event $k+1$, $k..(N+k)$ events are used). The paper by Evermann[14] proposes the use if RNN networks 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. Also, the paper [15] suggest developed a software suite for prediction tasks using deep learning and TensorFlow [5].

The most recent paper by Tax [36] uses RNN with LSTM cells to predict the next events. They achieve state of the art performance on most of the logs evaluated. In order to predict next event and time, they use one-hot encoding for event sequence, and few time features (such as the difference between time-stamps, time from midnight, time from the beginning of the week). Using these features they construct a vector to be fed into a 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 4).

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. We will exploit the state of the art architecture offered by [36] in order to build a model able to leverage A-priori knowledge.

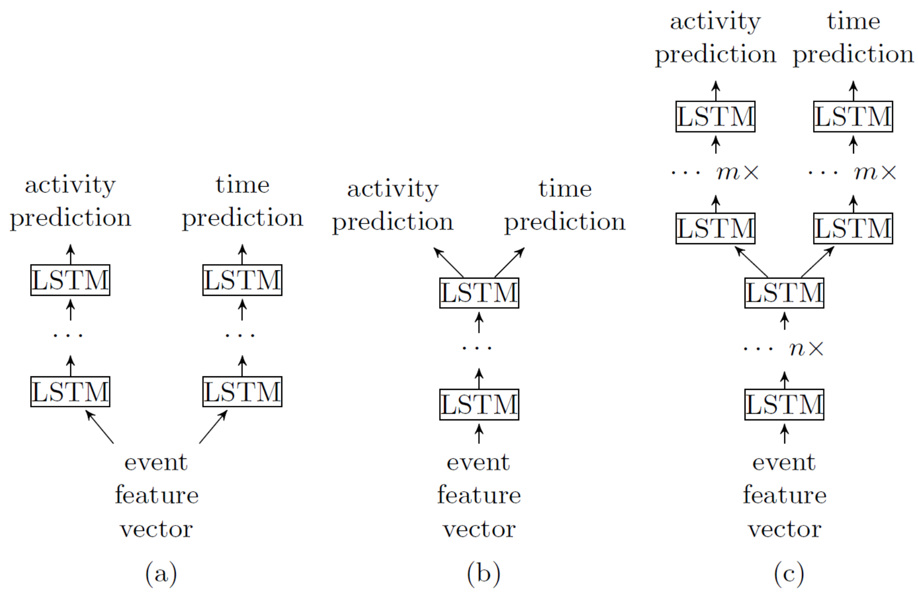

Fig. 4: Possible neural network architectures [36]. Single task layers (a), shared multitask layers (b), or n+m shared layers (c).

4 The Problem

Predictive process monitoring methods use past process executions, stored in event logs, in order to build predictive models that work at runtime to make predictions about the future. Among the different types of predictions about an ongoing case, such as the remaining time or the fulfillment 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-prior might be the weather information for an agricultural company that does the harvest. Knowledge about problems with diesel supplies can help a company to project on how their real processes concerning harvesting will unfold. For example assuming they are having delays with a supply in the middle of harvesting season, the 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. That could help the management to guide their choices to best account for the new circumstances.

Therefore, if this type of problems could be accurately solved it can bring much more benefits to the companies. For example, it can be also used for simulation purposes. For example a big insurance company, that deals with thousands of claims per day could decide that they need some changes such as cutting down jobs. Now they will have a capability to predict what will happen under the assumption of job cuts, and that could provide better insights. So besides classical qualitative methods of business process management that use BPMN models to analyze the changes my means of queuing theory or simulation, one would have one more instrument in the toolkit. This algorithm would enable a process owner to use some proposed changes as a-priori knowledge, and after predicting the continuations of the traces make further analysis.

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 an ongoing case be used to boost the 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 the current state of the art. *Trends of activities* means that we are interested in finding the suffix of the current ongoing process execution. *Ongoing case* means that we will have *prior knowledge* on the inference time. Therefore 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.

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):

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

For example, 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 [14,36] (see section 2.5 and 3.3). 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 retraining 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 called A-PRIORI technique that allows us to take into account a-priori knowledge expressed in terms of LTL rules (Section 5.2).

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 [36]. 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 works as follows: first, the current trace is analyzed in order to discover possible current cycles; and second, this information is used for preventing the prediction of a new repetition of the loop. More in detail:

1. For each prefix $p_k(\sigma) = \langle a_1 a_2 \ldots a_k \rangle$ of size k, the algorithm checks if there are j ($j >= 2$) consecutive occurrences of a cycle $c = \langle a_{c_1} \ldots a_{c_s} \rangle$, 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.

In simple words, the Nocycle technique looks for the current cycle. Figure 5 shows how the inference algorithm would choose the next event for prediction. Then figure 6 shows how the algorithm without Nocycle technique would choose next event, even though the cycle is present. If the technique applied, figure 7 shows that the probability to choose the event starting the cycle is diminished. After three times cycle is repeated (Figure 8) the algorithm predict a second most probable event, by this going out of the cycle.

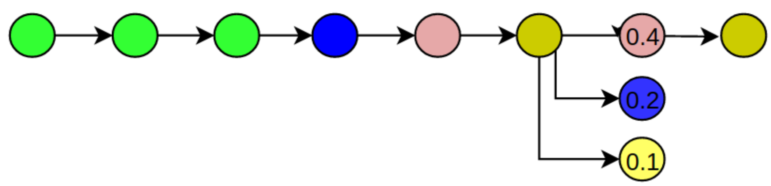

Fig. 5: How the next event is chosen

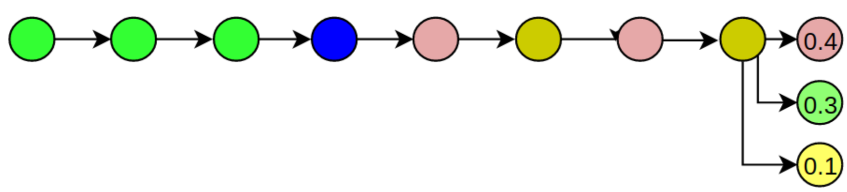

Fig. 6: How the next even standardly chosen (cycles present)

Algorithm 4 reports the pseudo-code of the Nocycle technique. Similarly to Algorithm 1 presented in Section 2.5, 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

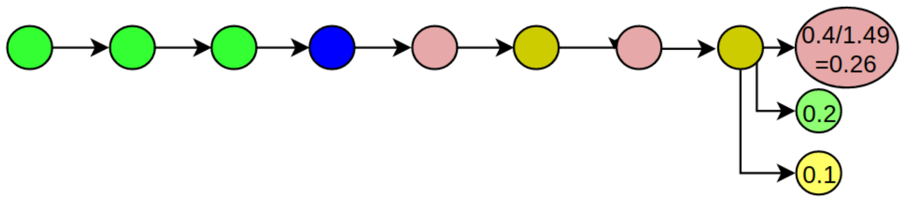

Fig. 7: Next event chosen after the e^2 applied for cycle size 2

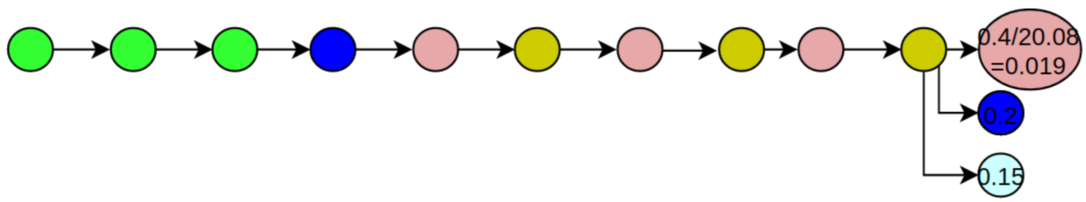

Fig. 8: Next event chosen after the e^2 applied for cycle size 3

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 $p_k(\sigma)$

function PredictSuffixNoCycle($p_k(\sigma)$, $lstm$, max)
    $k = 0$
    $trace = p_k(\sigma)$
    do
        $trace_{encoded}$ = encode($trace$)
        $next\_symbol\_prob$ = predictNextSymbols($lstm$, $trace_{encoded}$)
        $weak\_next\_symbol\_prob$ = weakenProb ($trace$, $next\_symbol\_prob$)
        $next\_symbol$ = getSymbol($weak\_next\_symbol\_prob$, $trace_{encoded}$)
        $trace = trace \cdot next\_symbol$
        $k = k + 1$
    while (not $next\_symbol == end\_symbol$) and ($k < max$)
    return trace
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; this can be done, because the step by step output of a recurrent neural network is basically a 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, independently of how many continuations of the trace are 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 leverages, on top of state-of-the-art LSTM techniques, the approach classically used in statistical sequence-to-sequence predictions in translation tasks [38], i.e., the *beamSearch* algorithm. The beamSearch works by the partial ordering of the probability trees, that is by exploring the search space, only taking the most promising branches. (As explained in section 2.7) In particular, we use the RNN architecture with LSTM cells and training system proposed in [36]. 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 algorithm (limiting parameter for a beam search inside of A-PRIORI procedure); (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.

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

```
function A-PRIORI(p_k(σ), K(σ), lstm, bSize, maxSize, max)
    h = 0
    prefixes = {p_k(σ)}
    while k ≤ max and not EMPTY(prefixes) do
        candidates_next = PREDICTPREFNEXTSYMBOLS(lstm, prefixes, bSize)
        top_candidates = TOPRANK(candidates_next, maxSize)
        for all candidate in top_candidates do
            if LAST_SYMBOL(candidate) <> end_symbol then
                PUSH(candidate, prefixes)
            else
                if CHECK(candidate, K) then
                    return candidate
                end if
            end if
        end for
        h = h + 1
    end while
end function
```

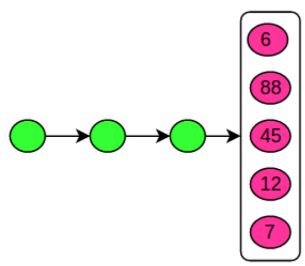

Fig. 9: Beam search based algorithm, step 1

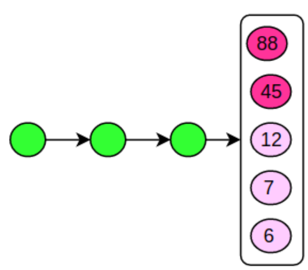

Fig. 10: Beam search based algorithm, step 2

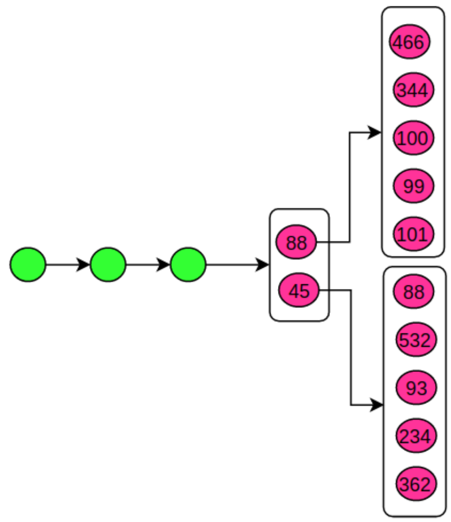

Fig. 11: Beam search based algorithm, step 3

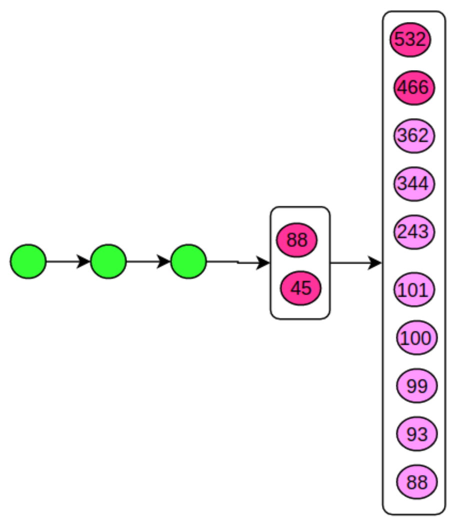

Fig. 12: Beam search based algorithm, step 4

To describe the algorithm we will use the graphics. Figure 9 shows in green the prefix of the trace, and the pink are the 5 predicted next events. Their scores are displayed inside the circles. In the next step (figure 10), the algorithm ranks the predicted events and chooses $bSize$ traces (in the exemplary case it's 2). The next step (figure 11), predicts the next events for all possible traces saved. Then these traces are combined together in one pool and also ranked with respect to the evaluation function(figure 12) and next $bSize$ events are chosen. After the check on the LTL formula approves the predicted sequence, and end symbol is predicted, the process stops, and the sequence is returned (every node keeps a link to its parent node, so reconstruction of the whole trace is not an issue).

Following pseudo code algorithm 5, 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 PREDICT-NEXTSYMBOLS procedure of Algorithm 1) and for each prefix $bSize$ 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 with 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).

The method A-PRIORI used in the evaluation is the initial algorithm combined with the NOCYCLE algorithm. To incorporate the NOCYCLE technique in the algorithm 5, line 5 method *predict-PrefNextSymbol* is updated to account for the cycle removal. Before returning *candidates_next* the current prefix is analyzed, and if a cycle is found, the next candidates with changed probabilities are returned.

[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 3: The event logs

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 neural networks. The LTL checker [2] for checking the compliance of traces to LTL rules is instead based on automata and written in Java [42][2]). 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 compare the proposed approaches and the most performant state-of-the-art approach [36] and 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],

[2] The LTL 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`

2012 [12], 2013 [33], 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 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). The logs containing less than 2 activities were filtered out, as they do not contribute to the evaluation. We briefly describe datasets in the following.

1. **Environment permit log** (*EnvLog*) contains data from a Dutch municipality[5]. The cases are the application process of the environment permits.
2. **Helpdesk** is a 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.
3. **BPIC11 log** is a real-life log taken from Dutch Academic Hospital. The cases of the log correspond to the Gynaecology department of the hospital. Phases (traces) consist of the diagnosis and treatment of a patient. Some attributes are repeated as the procedures could take place few times. As the log contains sensitive information it was anonymized.
4. **BPIC12 log**[6] This log is part of the BPI 2012 challenge, a real-life log, taken from a Dutch Financial Institute. The process of the event log is an application process for a personal loan or overdraft within a global financing organization.
5. **BPIC13 log**[7] is a log by Volvo IT in Belgium. The log consists of an incident and problem management from the Volvo "VINST" system.
6. **BPIC17 log**[8] is a log of a process taken from Dutch financial institute. In particular, the process we used is related to a loan application.

The characteristics of these logs are summarized in Table 3. For each log, we report the total number of traces, the number of activity

[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] http://www.win.tue.nl/bpi/doku.php?id=2013:challenge
[8] http://data.4tu.nl/repository/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

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.

Having a look at table 3, one can see the difference in characteristics of the logs. Environment Permit and BPIC11 are logs with a very long traces. They also have a large event alphabet size (299 and 355 respectively). Moreover BPIC15 has a big number of cycles in the log (5.05 per trace). Due to these characteristics, we expect these logs to have inferior results, as the in sparse solution space, machine learning algorithms generally have poor performance. BPIC12, BPIC13, BPIC17, and HelpDesk logs expectingly will have good performance because they contain lots of traces whilst having small alphabet size. Moreover, the traces BPIC12 and BPIC13 should be most benefited by the NOCYCLE technique, as they contain a number of cycles per trace.

6.2 Extraction of LTL rules

For testing with an a-priori algorithm, we need to establish the knowledge for every log. In order to provide unbiased knowledge and reproducible results, we decided not to handpick the LTL formulas but to establish a strict procedure for finding them instead.

Also, we wanted to test the algorithm with rules that differ in complexity, or "strength". With this in mind 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.

So we derived the a-priori knowledge on the traces of the testing set as follows:

1. We randomly selected 10% of traces of the testing set (in order not to find rules that are 100% satisfied in the whole testing set).
2. 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 (for example median of trace length of BPIC11 is 31, so we take prefixes of lengths 13,14,15, and 16). We derived all suffixes of these prefixes and we used the DeclareMiner ProM plug-in [24] to discover LTL rules satisfied in all those suffixes (Figure 13 shows the LTL extraction from DeclareMiner).

– 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 we composed them into a single conjunctive formula.

– 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.

The schematic form of the rules used in the evaluation is reported in Table 4, where - for the sake of readability - we replace the original activity names with single characters.

3. 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.

6.3 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 into two parts: a **training set** composed of the 67% of the whole event log and a **testing set**, composed of the remaining 33%.
2. The we extract LTL rules as described in section 6.2
3. we compared NOCYCLE and A-PRIORI[9] against a baseline provided by the technique presented in [36]. 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].

[9] 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).

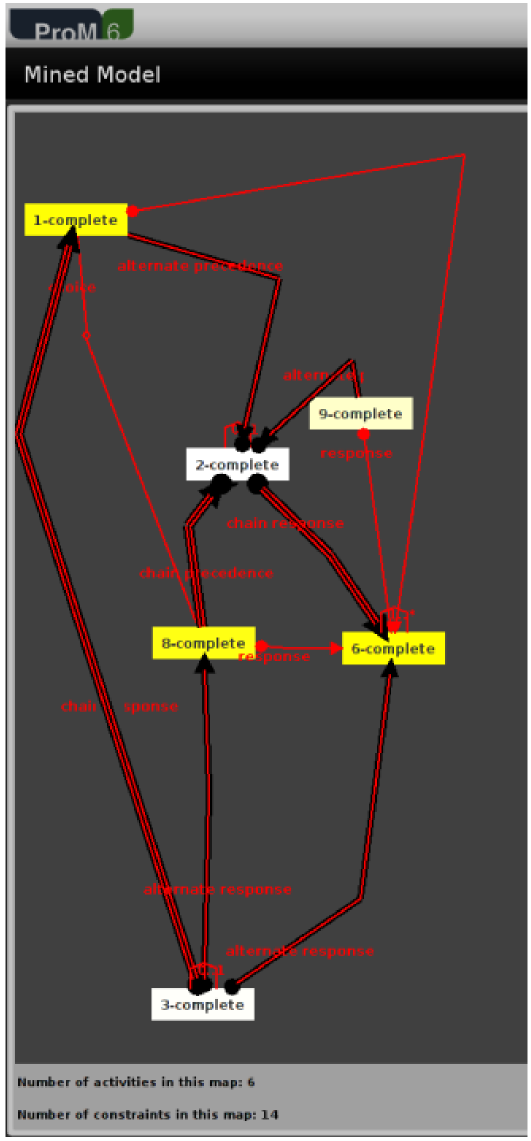

Fig. 13: Declare Miner in action

The experiments have been performed interchangeably on both 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. [36] as it has been shown to be the most performing ones

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 4: The a-priori knowledge.

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 5: Prediction results on the strong a-priori testing set

for facing the problem of predicting sequences of future activities.[10] The time required for training the LSTM neural 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.

6.4 Results and Discussion

Tables 5 and 6 report, for each event log, the performances of the two techniques we propose on the strong a-priori and weak a-priori testing sets. The results for both testing sets are compared with the baseline presented in [36]. 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 "Groundtruth"). The best average Damerau-Levenshtein similarity for each log is emphasized in gray. Column "Tested" reports the number of traces tested while column "Prefix" specifies the range of the prefix lengths used for the specific event log.

[10] 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.

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 6: Prediction results on the weak a-priori testing set

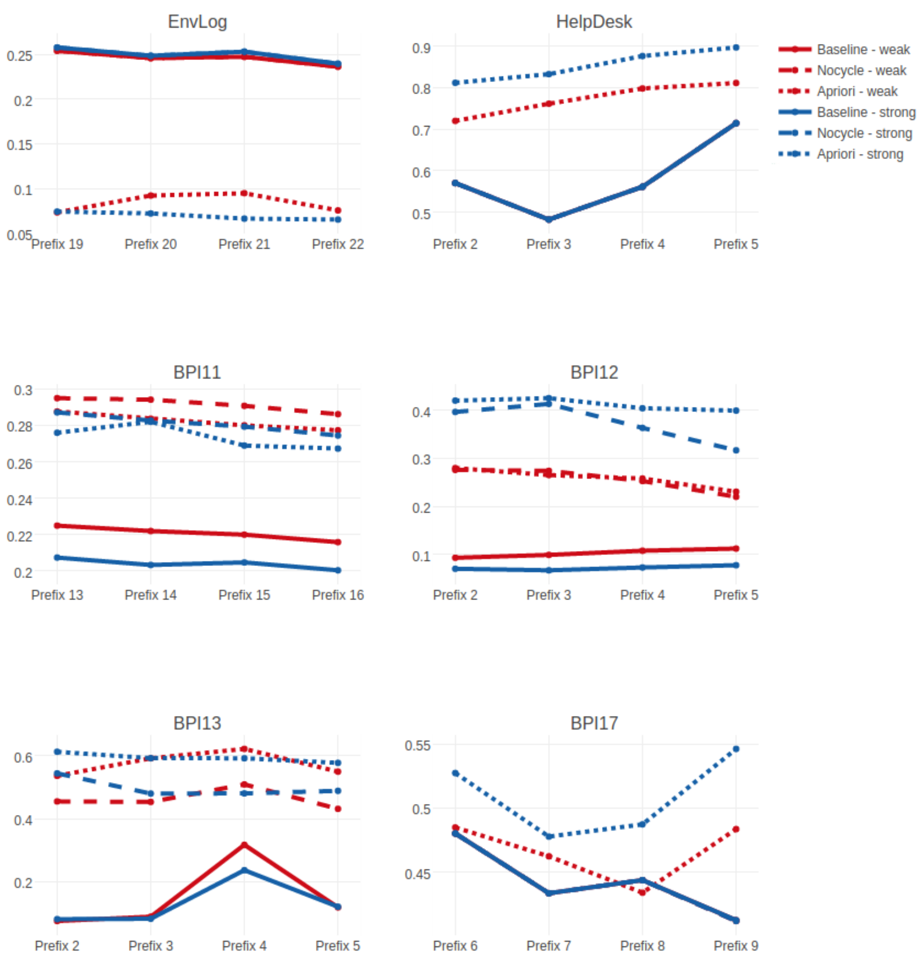

Fig. 14: Plots of prediction results

The tables and figure 14 (specifically, the figure shows the trend of the techniques performances with a change of a prefix. As one can see that sometimes lines on the plot perfectly overlap meaning that on some logs the weak and strong formulas do not make a difference) 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, activity labels are sparse with an unusually high number of labels with respect to the number of traces. Indeed, Table 3 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 the 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. Evermann, J., R.J., Fettke, P.: Xes tensorow - process prediction using the tensorflow deep-learning framework. In: CAiSE-2016 (2017)
16. 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)
17. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning, chap. Sequence Modeling: Recurrent and Recursive Nets, pp. 373–420. MIT Press (2016)
18. 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)

19. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
20. 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
21. 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
22. Lam, C., Ip, W., Lau, C.: A business process activity model and performance measurement using a time series {ARIMA} intervention analysis. Expert Systems with Applications 36(3, Part 2), 6986 – 6994 (2009), http://www.sciencedirect.com/science/article/pii/S0957417408006076
23. 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)
24. 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
25. 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)
26. 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)
27. 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)
28. 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)
29. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Time and activity sequence prediction of business process instances. CoRR abs/1602.07566 (2016)
30. Rogge-Solti, A., Weske, M.: Prediction of business process durations using non-markovian stochastic petri nets. Inf. Syst. 54(C), 1–14 (Dec 2015), http://dx.doi.org/10.1016/j.is.2015.04.004
31. 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
32. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining for delay prediction in multi-class service processes. Inf. Syst. 53, 278–295 (2015)
33. Steeman, W.: Bpi challenge 2013 (2013), https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07
34. 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)

35. 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
36. Tax, N., Verenich, I., Rosa, M.L., Dumas, M.: Predictive business process monitoring with lstm neural networks (2017), to appear in CAiSE 2017
37. 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)
38. 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)
39. 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
40. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, XESame, and ProM 6, pp. 60–75. Springer Berlin Heidelberg, Berlin, Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-17722-4_5
41. 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/
42. Westergaard, M.: Better algorithms for analyzing and enacting declarative workflow languages using ltl. In: Proceedings of the 9th International Conference on Business Process Management. pp. 83–98. BPM'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2040283.2040296