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

Chiara Di Francescomarino[1(✉)], Chiara Ghidini[1], Fabrizio Maria Maggi[2], Giulio Petrucci[1,3], and Anton Yeshchenko[2]

[1] FBK-IRST, Via Sommarive 18, 38050 Trento, Italy
{dfmchiara,ghidini,petrucci}@fbk.eu
[2] University of Tartu, Ulikooli 18, 50090 Tartu, Estonia
{f.m.maggi,anton.yeshchenko}@ut.ee
[3] University of Trento, Via Sommarive 14, 38050 Trento, Italy

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 paper, 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

1 Introduction

Predictive business process monitoring [19] is a research topic aiming at developing techniques that use event logs extracted from information systems in order to predict how ongoing (uncompleted) process executions (a.k.a. cases) will unfold up to their completion. A recent stream of work [12,13,23,28] has been focused on the provision of techniques able to predict the future path (continuation) of an ongoing case, a type of predictions that can be used to provide valuable input for planning and resource allocation. 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.

What motivates this paper 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.

In light of this motivation, in Sect. 5, we provide two techniques based on Recurrent Neural Networks with Long Short-Term Memory (LSTM) cells [16] 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 [28] 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 Sect. 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 (Sects. 5 and 6), the paper contains an introduction to some background notions (Sect. 2), a detailed illustration of the research problem (Sect. 4), related work (Sect. 3) and concluding remarks (Sect. 7).

2 Background

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

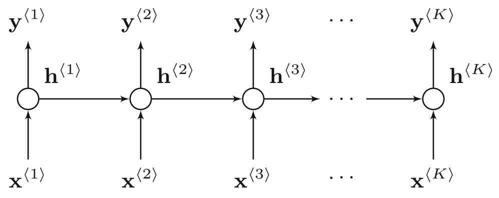
$\mathbf{y}^{\langle 1\rangle}$
$\mathbf{y}^{\langle 2\rangle}$
$\mathbf{y}^{\langle 3\rangle}$
...
$\mathbf{y}^{\langle K\rangle}$
$\mathbf{h}^{\langle 1\rangle}$
$\mathbf{h}^{\langle 2\rangle}$
$\mathbf{h}^{\langle 3\rangle}$
$\mathbf{h}^{\langle K\rangle}$
$\mathbf{x}^{\langle 1\rangle}$
$\mathbf{x}^{\langle 2\rangle}$
$\mathbf{x}^{\langle 3\rangle}$
...
$\mathbf{x}^{\langle K\rangle}$

Fig. 1. Recurrent Neural Network

2.1 Event Logs and Traces

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 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 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 [15]). 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 Eq. (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 Eq. (3)) explored in literature, Long Short-Term Memory (LSTM) [16] 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.

2.3 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 [28] relies on an encoding of activity sequences that combines 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 $A = \{a_{1_A}, \ldots a_{m_A}\}$ of all possible activities, an ordering function $idx : A \rightarrow \{1, \ldots, |A|\} \subseteq \mathbb{N}$ 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 $idx : A \rightarrow \{1, 2, 3\}$ 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 [28] 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:     $h = 0$
 3:     $trace = p_k(\sigma)$
 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:         $trace = trace \cdot next\_symbol$
 9:         $h = h + 1$
10:     while ($next\_symbol <> end\_symbol$) and ($h < max$)
11:     return $trace$
12: end function
```

2.4 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 Temporal Logic (LTL) rules [22]. 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), $\lozenge$ (future), $\square$ (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). $\lozenge\varphi$ indicates that φ is true sometimes in the future. $\square\varphi$ 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.

3 Related Work

The literature related to predictive business process monitoring can be roughly classified according to the type of predictions that is provided. A first group of works focuses on the time perspective. In [2], 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 [14], 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 use stochastic Petri nets to predict the remaining execution time of a process execution. In [20], 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 [25], queue theory is used 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 [6], 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 logs of past process executions. In [21], 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 [27], an approach for Root Cause Analysis through classification algorithms is presented.

A third group of prediction approaches predicts the outcome (e.g., the satisfaction of a business objective) of a case. In [19] a framework is introduced, which is able to predict the fulfillment (or the violation) of a boolean predicate in a running case, by looking at: (i) the sequence of activities already performed in the case; and (ii) the data payload of the last activity of the running case. The framework, which provides accurate results at the expense of a high runtime overhead, has been enhanced in [9] by introducing a clustering preprocessing step in which cases sharing a similar activity history are clustered together. A classifier for each cluster is trained with the data payload of the traces in the cluster. In [17], the authors compare different feature encoding approaches where traces are treated as complex symbolic sequences, that is, sequences of activities each carrying a data payload consisting of attribute-value pairs. In [29], unstructured information contained in text messages exchanged during process executions has been leveraged for improving the prediction accuracy.

The problem investigated in this paper falls into a fourth and last set of works, i.e., into the set of very recent efforts aiming at predicting the sequence of future activities given the activities observed so far. In [23], Polato et al. propose several techniques for predicting the remaining time and the sequence of future activities in an ongoing case using simple regression, regression with contextual information, and data-aware transition systems. Other approaches [12,13,28] make use of RNNs with LSTM cells. In particular, Evermann et al. [12,13] propose an RNN with two hidden layers trained with back propagation, while Niek et al. [28] leverage LSTM and an encoding based on activities and timestamps (illustrated in detail in Sect. 2.3) to provide predictions on the next activities and their timestamps. Differently from all these works, this paper investigates how to take advantage of possibly existing a-priori knowledge for making predictions on the sequence of future activities.

4 The Problem

Predictive business process monitoring methods use past process executions, stored in event logs, in order to build predictive systems that work at runtime to make predictions about the future. Among the different interesting and appealing types of predictions about the future of 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 a 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.

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 [12,13,28]. 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 (Sect. 5.1). We then describe A-priori, an algorithm that allows us to take into account a-priori knowledge expressed in terms of LTL rules (Sect. 5.2). In both cases, we use the RNN architecture with LSTM cells and training system proposed in [28], while we extend and enhance the prediction phase. The A-priori 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 [28]. 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 2 reports the pseudo-code of the NOCYCLE technique. Similarly to Algorithm 1 presented in Sect. 2.3, 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.

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; (ii) we rank them according to the likelihood of the prediction; and (iii) we select the first prediction that 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.

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 [30], i.e., the *beamSearch* algorithm. The beamSearch is a heuristic algorithm based on graphs that explores the search space by expanding only the most promising branches. Then, in the testing

Algorithm 2. NOCYCLE extension for predicting the suffix of $p_k(\sigma)$

```
1: function PREDICTSUFFIXNOCYCLE($p_k(\sigma)$, $lstm$, $max$)
2:     $h = 0$
3:     $trace = p_k(\sigma)$
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:         $trace = trace \cdot next\_symbol$
10:        $h = h + 1$
11:    while ($next\_symbol <> end\_symbol$) and ($h < max$)
12:    return trace
13: end function
```

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 3 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 maximum number of next symbols predicted by the LSTM model and used to construct the possible predicted suffixes at each iteration; (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 3. A-PRIORI algorithm for predicting the suffix of $p_k(\sigma)$

```
1: function A-PRIORI ($p_k(\sigma)$, $\mathcal{K}(\sigma)$, $lstm$, $bSize$, $maxSize$, $max$)
2:     $h = 0$
3:     $prefixes = \{p_k(\sigma)\}$
4:     while ($h \leq max$) and (not ISEMPTY($prefixes$)) do
5:         $candidates\_next$ = PREDICTPREFNEXTSYMBOLS($lstm$, $prefixes$, $bSize$)
6:         $top\_candidates$ = TOPRANK($candidates\_next$, $maxSize$)
7:         EMPTY($prefixes$)
8:         for all $candidate$ in $top\_candidates$ do
9:             if LAST_SYMBOL($candidate$) <> $end\_symbol$ then
10:                PUSH($candidate$, $prefixes$)
11:            else
12:                if CHECK($candidate$, $\mathcal{K}$) then
13:                    return $candidate$
14:                end if
15:            end if
16:        end for
17:        $h = h + 1$
18:    end while
19: end function
```

Intuitively, the algorithm iterates over a priority queue of prefixes, which is initialized with the input prefix $p_k(\sigma)$ (line 3) and is used for regulating the number of branches to be explored. For each prefix in $prefixes$, $bSize$ possible next activities are predicted using the model $lstm$ 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 8), if the last symbol predicted is not the end symbol, the trace is added to $prefixes$ (line 10). Otherwise, if the trace is complete, the algorithm checks if it is compliant to the LTL rules in $\mathcal{K}(\sigma)$ (line 12). In this case, the trace is returned (line 13). 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 2 and 3 (and their combination) have been implemented in Python 2.6. In particular, the Keras [5] and TensorFlow [3] libraries have been used for neural networks. The LTL checker for checking the compliance of traces with respect to LTL rules is instead based on automata and written in Java. The Py4J library has been used as a gateway to access Java code from Python. The full source code is available on github at https://github.com/yesanton/ProcessSequencePrediction.

6 Evaluation

In this section, we provide an evaluation of our predictive business process monitoring techniques based on a-priori knowledge. In detail, 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 accuracy of the predictions; and (ii) whether the combination of Nocycle with A-priori, 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 Challenge (BPIC) 2011 [1], 2012 [10], 2013 [26], and 2017 [11], 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 [4] (*EnvLog* for short in this paper), and another containing cases from a ticketing management process of the help desk of an Italian

[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.

Table 1. The event logs

Log	**#Tr.**	**#Act.**	**avg-TL**	**avg-CR**	**Spars.**
EnvLog	937	381	41.562	0.14	0.3191
HelpDesk	3804	9	3.6	0.22	0.0024
BPIC11	911	424	54.168	5.05	0.4654
BPIC12	9 658	6	7.5	1.35	0.0006
BPIC13	7 554	13	8.675	1.45	0.0017
BPIC17	31 508	26	17.826	0.46	0.0008

software company (*Helpdesk*[2] for short). Note that all the logs have been filtered. In particular, *BPIC12*, *BPIC13*, *EnvLog* and *HelpDesk* are the ones used in [28], in order to ease the comparison of our techniques with the state-of-the-art. Similarly, *BPIC11* and *BPIC17* have been filtered by removing outlier traces with respect to the average trace length.

The characteristics of these logs are summarized in Table 1. 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 this paper, we adopted the following procedure. For each event log:

1. We divided the event log in two parts: a **training set** composed of 67% of traces of the whole event log used for building the LSTM models and a **testing set** composed of the remaining 33% used for testing the predictions of suffixes.
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. We used the DeclareMiner ProM plug-in [18] to discover LTL rules satisfied in all these traces. 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[3] rules of type $\Diamond A$ and 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

[2] https://data.mendeley.com/datasets/39bp3vv62t/1.

[3] The number of rules selected has been determined empirically to allow them to be satisfied in around 50% of the traces of the testing set.

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 in the testing set. The schematic form of the rules used in the evaluation is reported in Table 2, 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 satisfying strong and weak a-priori knowledge.

3. From each trace in the testing sets, 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. Then, we compared NOCYCLE and A-PRIORI* against a baseline provided by the technique presented in [28], when predicting the suffixes of these prefixes.[4] 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 [7].

Table 2. The a-priori knowledge

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	$\Box(u \rightarrow \Diamond v) \wedge \Diamond u$	$\Diamond u$

The experiments have been performed both on a GPU Tesla K40c and on a conventional laptop CPU on Code i5. As for the LSTM training settings we used the ones identified by Tax et al. [28] as the most performing ones for facing the problem of predicting sequences of future activities.[5] The time required for training the LSTM models is about 2 min per epoch using the GPU and 15 min 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.

6.3 Results and Discussion

Tables 3 and 4 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

[4] 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).

[5] 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.

both testing sets are compared with the baseline presented in [28]. For each log, we provide the average Damerau-Levenshtein similarity between the predicted sequence (in square brackets, its average length) and the ground truth (in column 5 its average length). 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.

Table 3. Prediction results on the strong a-priori testing set

Log	Baseline		Nocycle		A-priori*		Groundtruth	Tested	Prefix
EnvLog	0.250	[17.40]	0.250	[17.4]	0.070	[95.00]	29.40	80	19 – 22
HelpDesk	0.551	[1.44]	0.551	[1.44]	0.816	[2.53]	3.00	576	2 – 5
BPIC11	0.204	[199.00]	0.281	[199.00]	0.276	[196.3]	117.11	144	13 – 16
BPIC12	0.071	[47.07]	0.387	[6.86]	0.408	[9.02]	10.95	1 548	2 – 5
BPIC13	0.116	[100.80]	0.502	[14.71]	0.516	[17.75]	7.15	3 209	2 – 5
BPIC17	0.448	[11.78]	0.448	[11.78]	0.439	[21.90]	16.01	10 153	6 – 9

Table 4. Prediction results on the weak a-priori testing set

Log	Baseline		Nocycle		A-priori*		Groundtruth	Tested	Prefix
EnvLog	0.246	[18.22]	0.246	[18.22]	0.068	[95.00]	31.31	108	19 – 22
HelpDesk	0.551	[1.44]	0.551	[1.44]	0.816	[2.53]	3.00	576	2 – 5
BPIC11	0.220	[199.00]	0.292	[199.00]	0.287	[197.01]	112.66	450	13 – 16
BPIC12	0.100	[48.08]	0.263	[6.81]	0.273	[7.84]	8.33	3 179	2 – 5
BPIC13	0.130	[95.19]	0.459	[14.92]	0.476	[7.45]	5.85	4 364	2 – 5
BPIC17	0.448	[11.78]	0.448	[11.78]	0.424	[24.92]	16.01	10 153	6 – 9

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* performs worse on logs EnvLog and 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 1 shows that the ratio between the number of activity labels and the number of traces (column 6) 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 this paper, 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, by 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 leveraging 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; (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 ones discussed in [8,9].

Acknowledgments. This research has been partially carried out within the Euregio IPN12 KAOS, which is funded by the "European Region Tyrol-South Tyrol-Trentino"(EGTC) under the first call for basic research projects.

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., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Inf. Syst. **36**(2), 450–475 (2011)
3. 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). software available from tensorflow.org. http://tensorflow.org/
4. Buijs, J.: Environmental permit application process ("wabo"), coselog project - municipality 4 (2014). doi:10.4121/uuid:e8c3a53d-5301-4afb-9bcd-38e74171ca32
5. Chollet, F.: Keras (2015). https://github.com/fchollet/keras
6. Conforti, R., de Leoni, M., La Rosa, M., van der Aalst, W.M.P.: Supporting risk-informed decisions during business process execution. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 116–132. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38709-8_8
7. Damerau, F.J.: A technique for computer detection and correction of spelling errors. Commun. ACM **7**(3), 171–176 (1964)
8. Di Francescomarino, C., Dumas, M., Federici, M., Ghidini, C., Maggi, F.M., Rizzi, W.: Predictive business process monitoring framework with hyperparameter optimization. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 361–376. Springer, Cham (2016). doi:10.1007/978-3-319-39696-5_22

9. Di Francescomarino, C., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Trans. Serv. Comput. **PP**(99), 1–18 (2016)
10. van Dongen, B.: Bpi challenge 2012 (2012). doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
11. van Dongen, B.: Bpi challenge 2017 (2017). doi:10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
12. Evermann, J., Rehse, J.-R., Fettke, P.: A deep learning approach for predicting process behaviour at runtime. In: Dumas, M., Fantinato, M. (eds.) BPM 2016. LNBIP, vol. 281, pp. 327–338. Springer, Cham (2017). doi:10.1007/978-3-319-58457-7_24
13. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems (2017)
14. Folino, F., Guarascio, M., Pontieri, L.: Discovering context-aware models for predicting business process performances. In: Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F. (eds.) OTM 2012. LNCS, vol. 7565, pp. 287–304. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33606-5_18
15. Goodfellow, I., Bengio, Y., Courville, A.: Sequence Modeling: Recurrent and Recursive Nets. In: Deep Learning, pp. 373–420. MIT Press, Cambridge (2016)
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
17. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 297–313. Springer, Cham (2015). doi:10.1007/978-3-319-23063-4_21
18. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31095-9_18
19. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 457–472. Springer, Cham (2014). doi:10.1007/978-3-319-07881-6_31
20. 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, SRII 2012, pp. 313–322. IEEE Computer Society, Washington, DC (2012)
21. 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. In: Rosa, M., Soffer, P. (eds.) BPM 2012. LNBIP, vol. 132, pp. 211–216. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36285-9_22
22. 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)
23. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Time and activity sequence prediction of business process instances. CoRR abs/1602.07566 (2016)
24. Rogge-Solti, A., Weske, M.: Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 389–403. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45005-1_27
25. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining for delay prediction in multi-class service processes. Inf. Syst. **53**, 278–295 (2015)

26. Steeman, W.: Bpi challenge 2013 (2013). doi:10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07
27. Suriadi, S., Ouyang, C., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Root cause analysis with enriched process logs. In: La Rosa, M., Soffer, P. (eds.) BPM 2012. LNBIP, vol. 132, pp. 174–186. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36285-9_18
28. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 477–492. Springer, Cham (2017). doi:10.1007/978-3-319-59536-8_30
29. Teinemaa, I., Dumas, M., Maggi, F.M., Di Francescomarino, C.: Predictive business process monitoring with structured and unstructured data. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 401–417. Springer, Cham (2016). doi:10.1007/978-3-319-45348-4_23
30. Tillmann, C., Ney, H.: Word reordering and a dynamic programming beam search algorithm for statistical machine translation. Comput. Linguist. **29**(1), 97–133 (2003)