Embedding Graph Convolutional Networks in Recurrent Neural Networks for Predictive Monitoring

Efrén Rama-Maneiro, Juan C. Vidal, and Manuel Lama

Abstract—Predictive monitoring of business processes is a subfield of process mining that aims to predict, among other things, the characteristics of the next event or the sequence of the next events. Although multiple approaches based on deep learning have been proposed, mainly recurrent neural networks and convolutional neural networks, none of them really exploit the structural information available in process models. This paper proposes an approach that simultaneously learns spatio-temporal information from both the event log and the process model by combining recurrent neural networks with graph convolutional networks. Thus, common patterns from process models, such as loops or parallels, can be learned while avoiding overwriting information during the encoding phase. An experimental evaluation of real-life event logs shows that our approach is more consistent and outperforms the current state-of-the-art approaches.

Index Terms—Process mining, Predictive business monitoring, Deep Learning, Graph Neural Networks, Recurrent Neural Networks

1 Introduction

Process mining [1] is a discipline that aims to describe the future and past behavior of a business process, given the information available in an event log. Multiple analyses can be applied to event logs, such as discovering a process model as an abstract representation of the underlying business process (*process discovery*), improving a process model using the information from the event log (*process enhancement*), or comparing a process model with an event log to check its degree of conformance (*process conformance*) [1].

However, it not only focuses on the description of the current process behavior but also on predicting its future behavior. Thus, predictive monitoring is a subfield of process mining concerned with forecasting how an ongoing case is going to unfold in the future [2]. These predictions may involve information such as the next activity or sequence of activities, when the next event will happen, or how much time is left until the end of the case. Many machine learning techniques have been applied to predictive monitoring, although approaches based on deep learning are the ones that perform better [3]. Recurrent Neural Networks (RNNs), are the most popular in this domain due to the sequential nature of traces in processes [4]. However, autoencoders [5], [6], generative adversarial networks (GANs) [7], convolutional neural networks (CNNs) [8], [9], or other types of RNNs [10], [11], [12] have also been used.

- *Efrén Rama-Maneiro, Juan C. Vidal, and Manuel Lama are with the Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain. Juan C. Vidal is also with the Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Galicia, Spain. email: {efren.rama.maneiro, juan.vidal, manuel.lama}@usc.es*

Manuscript received April 22, 2023 revised X

Almost all deep learning approaches in predictive monitoring build a predictive model exclusively from the information available in the traces of the processes [5], [6], [7], [8], [9], [10], [11], [13], [14], [15], [16], [17]. These approaches disregard the explicit structural information available in the process models, i.e., behavioral patterns such as loops and parallels may be missing since the neural networks might not be able to detect them with only the trace information. On the other hand, the approaches that rely on process models as input [18], [19], [20] obtain information from the connectivity between the model activities [19], [20] or by performing a token replay over its model [18]. However, these model-based approaches are unable to fully leverage the structural and temporal information available in the event log. They disregard the order in which the events appear in the prefix, are subjected to overwrite features if any activity appears repeated in the prefix, and most of them [19], [20] rely on Directly Follows Graph (DFGs) process models, which are less expressive than Petri nets [21].

In this paper, we hypothesize that the performance of predictive models can be improved by combining the information from both the traces and the process model and leveraging the information on how the execution of the process model behaves over time. Thus, we propose to capture this execution using graph neural networks (GNNs) and RNNs by taking into account the full sequence of states and not only the state of the replay previous to the prediction. This should facilitate the detection of some behavioral patterns common in process models, such as loops and parallels, in addition to time-related behavioral patterns. Parallels, in which a set of activities might appear in any order in a trace, are harder to detect without considering the process model, so knowing their presence beforehand could ease the training phase of the neural network. Regarding loops, the neural network might benefit from

knowing beforehand where the loop is going back, which activities compose the loop, and whether inner loops exist. Moreover, using the process model as an additional input could help the neural network focus more on other event log information since the relationships between the activities are already explicitly available from the process model. We validated our approach using ten publicly available datasets against the main state-of-the-art approaches, showing that our approach improves the accuracy of predictions in almost all cases.

In summary, the main contributions of our graph recurrent neural model are as follows:

- A neural model that simultaneously learns spatio-temporal information from both the event log and the process model by combining recurrent neural networks and graph convolutional neural networks, which, unlike other state-of-the-art approaches, learns the process model structure explicitly. Furthermore, the proposed model not only considers the structural information of the model but explicitly takes into account the information extracted from the events of the prefix.
- A process model encoding that allows learning the dependencies between the loops and parallels, while retaining memory efficiency during training. This encoding, unlike other encodings proposed in the state of the art, retains the full information of the prefix, without overwriting information during the encoding phase.
- An evaluation of the proposal on a wide variety of publicly available event logs, including a comparison with 10 state-of-the-art proposals, an ablation study on both the structural components of the architecture and the information extracted from the prefix, an analysis of the approach performance w.r.t. the presence of loops in the event logs, and an analysis of the hyperparameter effect on the approach results.

This paper is structured as follows, Section 2 shows the state of the art in deep learning-based techniques for predictive monitoring, Section 3 shows some background needed to build the approach, Section 4 describes the proposed approach, Section 5 shows the evaluation of the proposed approach in real-life event logs, and Section 6 highlights the conclusions of the paper and future work.

2 Related work

The first works in predictive monitoring with deep learning heavily relied on RNNs to learn predictive models. For example, [14] use LSTM neural networks to predict both the next activity and the next timestamp in an ongoing process instance. The activities are encoded as a one-hot vector, and they also consider time features such as the time passed since the previous event or the beginning of the case. [13] also aims to predict the next activity by encoding the activity through embeddings instead of a one-hot vector, thus reducing the dimensionality of the input.

Other types of architectures based on recurrent neural networks have also been explored. [11] uses a Differentiable Neural Computer [22], which is a type of neural network that has an external memory to enhance the representation of longer-term dependencies in a sequence. They define an encoder-decoder that is trained to predict either the next activity or the next timestamp. [23] also uses an encoder-decoder network, but they rely on an attention mechanism to take into account every hidden state of the LSTM that learns from the input sequence. These kinds of architectures may yield better results at the expense of increasing the complexity of the network, or more training difficulties than simpler types of RNN.

Some predictive monitoring approaches propose novel ways to encode the attributes of the event log. [15] clusters the resources available in the event log into roles and learn an LSTM that simultaneously predicts the next activity, next timestamp, and next role. [10] also clusters the events based on their attributes and uses these clustering labels as additional information in a recurrent neural network. The main advantage of these types of encodings is reducing the complexity of the input by grouping attributes in fixed categories, however, at the risk of discarding potentially relevant information.

Some other approaches are based on CNNs. [9] uses CNNs to predict the next activity. They rearrange the prefixes in a grid-like fashion using an encoding in which, for each event, they count the number of occurrences present in the prefix. These prefixes are fed to a two-dimensional CNN, which is used to predict the next activity. [8] also employs CNNs by adapting the Inception model [24] to predict the next activity, outperforming LSTMs in some event logs. In [12], Gated Convolutional Neural Networks [25] and Key-Value-Predict [26] Attention Networks are introduced. The former combines convolutional networks with a gating mechanism, similar to the one used in LSTMs and GRUs, while the latter tries to learn the correlation between pairs of elements that belong to the input sequence by means of an attention mechanism. The main advantage of convolutional neural networks is that they are computationally more efficient than RNNs. Nevertheless, they are more complex to design since they have more variation, such as whether and how to stack pooling layers and what size of the convolution filter shall be used. In predictive monitoring, RNNs seem to outperform CNNs due to this cause [4].

Furthermore, some approaches are testing novel architectures for predictive monitoring. [7] rely on GANs [27] to predict the next activity and timestamp of an ongoing process instance, hypothesizing that this type of neural network would alleviate the need for high amounts of training data. [16] are focused on implementing Transformers [28] architectures for predictive monitoring, a type of neural network that relies exclusively on attention mechanisms. Thus, they can avoid predictive performance degradation when the RNN faces long sequences and improves the learning phase training and inference speed. Even though the results of these approaches look promising, there are still doubts about whether they can perform reliably across a wide variety of datasets, and, even so, they do not use the potential information available in process models.

Finally, some works rely on explicit process models to help encode the prefixes. [18] builds feature vectors by performing a token replay of a prefix over a process model. These feature vectors include the information about the

most recent activation on a Petri net, the time of activation, which is a decay function, the number of tokens, and the attributes available in the event log. These vectors are then fed to a deep feed-forward neural network. In [20], the usage of Gated Graph Neural Networks is explored. Their adjacency matrix can be based on the events of a prefix, the activities of the prefix, or the Directly Follows Graph (DFG) extracted from the event log. The edges in their graph convey information about the prefix. [19] investigates the usage of GCNs for predicting the next activity. Their adjacency matrix is based on the DFG, and they focus on the differences in various methods of encoding this adjacency matrix (binary, weighted, with a Laplacian transform, ...).

However, the model-based approaches are still lacking in terms of taking full advantage of the structural and temporal information available in the event log. These approaches disregard the order in which the events appear in the event log. For example, [18] only takes into account this order by means of a decay function associated with each of the places of the Petri net, only accounting for its most recent activation, while [19] and [20], only consider the event order using time features extracted from the event log, such as the time since the last event. Structurally speaking, [18] relies on MLPs for their architecture, so the process model is not explicitly considered. Both [19] and [20] rely on GNNs but they use DFGs as their process model, which is less expressive than Petri nets. Furthermore, every model-based approach only retains the most recent activation for an activity or place, so they are prone to overwrite information when a loop occurs.

Our approach relies on embeddings to encode the categorical features, similarly to [13], because we use the same attributes as in [4] and some logs have a high number of different attributes; therefore, using a one-hot encoding would increase the memory consumption dramatically [4]. Furthermore, our approach both learns structural properties explicitly from a process model using GCNs and learns temporal information from the order of the events within the trace. Thus, unlike the other model-based approaches from the state of the art, we capture the whole time dimension from the prefix, without being affected by overwriting features when an activity is repeated, as is the case with [18], [19], [20]. Moreover, our approach is based on Petri nets, which are more expressive than DFGs. In addition, the latter is unable to represent parallels or distinguish OR from XOR splits [21]. Therefore, we can capture more complex relationships between the events, unlike other model-based approaches.

3 PRELIMINARIES

In this section, we present the main concepts needed to understand our approach for predicting the next activity of a running case. A symbol table is highlighted in TABLE 1, for ease of reference.

3.1 Definitions

Definition 1 (event). Let AC be the universe of activities, C the universe of cases, TI the time domain, and $D_1, \ldots, D_m$ the universes of each of the attributes of the events of the event log, with $m \geq 0$. An event $e \in E$ is a tuple $(a, c, t, d_1, \ldots, d_m)$ where $a \in AC$, $c \in C$, $t \in TI$ and $d_i \in \{D_i \cup \epsilon\}$ with $i \in [1, m]$ and ϵ being the empty element.

Definition 2 (projection function). Let π_{AC}, π_C, π_{TI}, and π_{D_i} be functions that map an event to an activity, a case identifier, a timestamp, and an attribute, that is, $\pi_{AC}(e) = a$, $\pi_C(e) = c$, $\pi_{TI}(e) = t$, and $\pi_{D_i}(e) = d_i$.

Definition 3 (trace). Let S be the universe of traces, then a trace $\sigma \in S$ is a non-empty sequence of events $\sigma = \langle e_1, \ldots, e_n \rangle$, which holds that $\forall e_i, e_j \in \sigma; i, j \in [1, n] : j > i \wedge \pi_C(e_i) = \pi_C(e_j) \wedge \pi_{TI}(e_j) \geq \pi_{TI}(e_i)$ where $|\sigma| = n$.

Definition 4 (event log). An event log is a set of traces, $B = \{\sigma_1, \ldots, \sigma_l\}$ such as $B = \{\sigma_i | \sigma_i \in S \wedge i \in [1, b]\}$ where $|B| = b$.

Let TABLE 2 be an example of an event log from a loan application process [29]. In this event log, there are four different activities, each representing the task performed by a resource to achieve a goal [30]. The sequence of activities belonging to the same trace identifier is called a trace, and a partial trace, which is used as an input to a predictive model, is called a prefix. In this example, there are two different traces: "214364" and "173712". Note that the activities in a trace must appear ordered chronologically according to their execution time, represented by the column "timestamp" in this case. Event logs can optionally have additional information, such as the resource that executes each activity.

Trace ID	Activity	Timestamp	Resource
173712	W_Afhandelen leads (**WAL**)	01/10/2011 08:45:13	Gordon
173712	W_Completeren aanvraag (**WCA**)	02/10/2011 09:26:51	Alex
173712	W_Completeren aanvraag (**WCA**)	03/10/2011 10:39:04	Barney
173712	W_Completeren aanvraag (**WCA**)	04/10/2011 11:51:33	Adrian
214364	W_Afhandelen leads (**WAL**)	06/10/2011 09:10:22	Joseph
214364	W_Completeren aanvraag (**WCA**)	06/10/2011 15:13:22	Joseph
214364	W_Nabellen offertes (**WNO**)	08/10/2011 10:48:11	Enrico
214364	W_Valideren aanvraag (**WVA**)	08/10/2011 11:10:32	Harambe

TABLE 2: Potential example from the BPI-2012-W-C event log.

Symbol	Description
AC	Universe of activities ($a \in AC$)
C	Universe of cases ($c \in C$)
TI	Time domain ($t \in TI$)
D_n	Universe of the attribute n ($d_n \in D_n$)
π_X	Projection function to a universe X
S	Universe of traces ($\sigma \in S$)
PT	Petri net
P	Set of places from a Petri net ($p \in P$)
T	Set of transitions from a Petri net ($t \in T$)
F	Set of directed arcs from a Petri net
V	Set of vertices of the place graph
E	Set of edges of the place graph
A	Adjacency matrix of a graph
I_N	Identity matrix
N	Number of nodes of a graph
H	Hidden size
Q	Set of node features
X	Feature matrix (unspecified)
R	Set of attribute features
L	Length of the longest trace of the event log
$\oplus$	Concatenation operator

TABLE 1: List of symbols used in the paper

Definition 5 (prefix). Let σ be a trace such as $\sigma = \langle e_1, \ldots, e_n \rangle$ and $k \in [1, n]$ be any positive integer. The event prefix of length k, hd^k can be defined as follows: $hd^k(\sigma) = \langle e_1, \ldots, e_k \rangle$. The activity prefix can be defined as the application of π_{AC} to the whole event prefix, being $\pi_{AC}(hd^k(\sigma)) = \langle \pi_{AC}(e_1), \ldots, \pi_{AC}(e_k) \rangle$, respectively.

In this paper, we focus on predicting the next activity of a given prefix. Formally:

Definition 6 (next activity prediction). Let $hd^k(\sigma)$ be an event prefix such as $hd^k(\sigma) = \langle e_1, \ldots, e_k \rangle$, and e' be a predicted event by a function Ω, then, the next activity prediction problem can be defined as learning a function Ω_{AC} such as $\Omega_{AC}(hd^k(\sigma)) = \pi_{AC}(e'_{k+1})$.

In this paper, we leverage information from process models using deep learning. Our business process models are represented by a place/transition Petri net [31] because it allows a richer representation of the behavior of a business process than other approaches such as the DFG. A Petri net is a bipartite graph composed of two types of nodes: places, which can contain tokens, and transitions. More formally, a Petri net can be defined as follows:

Definition 7 (Petri net). A Petri net is a tuple $PT = (P, T, F)$ where:

- P is a finite set of places.
- T is a finite set of transitions.
- $P \cap T = \emptyset$
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs.

Transitions can be classified into *observable transitions*, which are associated with a process activity, and *silent transitions*, which are related to control flow routing. Thus, the order of the execution of the activities within a prefix is modeled through the flow of the tokens across the network. The places contain the tokens that are produced/consumed by transitions, while the marking of a Petri net is the number of tokens in a given place. These tokens are produced and consumed when a transition is executed. When this happens, a token is removed from each input place of the transition and is inserted in every output place of it. This process can only be performed when a transition is enabled, i.e., when the input places of a transition have at least one token.

Recall Fig. 1, which shows the execution semantics of a process model represented as a Petri net mined from the BPI-2012-W-C from TABLE 2. In this model, places are represented as circles, observable and silent transitions are represented as white and black rectangles, respectively, and red transitions represent each fired transition for each event replayed in the model. Thus, the replay of the trace with the case identifier "214364" in the process model would produce the execution semantics depicted in the figure.

Our model reduces the Petri net of the process model to a *place graph* that is easier to handle in our deep learning-based solution.

Definition 8 (Place graph). A place graph is a directed graph $G = (V, E)$ with vertices V and edges E that represents a Petri net $PT = (P, T, F)$ where:

- $v \in V \iff p \in P$.

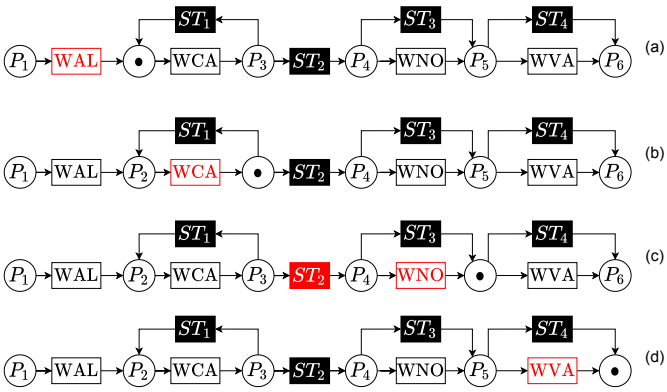

Fig. 1: Execution semantics from the Petri net mined from the event log of TABLE 2.

- Each edge $e \in E$ connects two vertices $v \in V$ if and only if a transition interconnects two places, i.e., let $t \in T$; $p_1, p_2 \in P$; and, $v_1, v_2 \in V$; then $(v_1, v_2) \in E \rightarrow (p_1, t) \in F \wedge (t, p_2) \in F$.

The place graph is represented in our approach by a binary adjacency matrix, i.e., $A \in \mathbb{R}^{|P| \times |P|}$. Modeling the Petri net as a place graph has two advantages. First, the interactions between the activities of a Petri net can be represented only by the tokens consumed or produced in the places because each event is associated with a snapshot of the execution state of the Petri net. Second, the memory consumption is reduced by approximately a factor of two, which is essential since the dimensions of every matrix related to the graph depend on the number of its nodes.

3.2 Graph Recurrent Neural Networks

GNNs are a type of neural network that operates on the graph structure, capturing dependencies between each node of a graph. Graph convolutional networks (GCN) [32] are a type of GNN architecture that is defined as follows:

$$g_\theta \star x \approx \theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x) \tag{1}$$

Where I_N is the identity matrix, A is the adjacency matrix of the graph, and D is the degree matrix of the graph. Equation 1 can lead to exploding/vanishing gradients, so the *renormalization trick* $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ is introduced in [32] by adding self-loops to the graph, i.e, $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Thus, we achieve the following GCN operator:

$$GCN(A, X) = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \tag{2}$$

Where Θ is a matrix of learnable filter parameters. This model is unsuitable for predictive monitoring for two reasons. First, it does not take into account directionality in the input place graph due to the symmetric normalization $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$. Second, it does not fully consider the information available in the whole sequence of execution states related to the events since it can only compute the last state. Thus, this model is prone to the problem of loop overwriting (detailed in section 4.2) and, by extension, to the loss of information contained in the prefix.

To alleviate the first issue, we modify the GCN operator to use the random walk normalized Laplacian with the renormalization trick. Thus, the directionality of the graph could be taken into account as follows [33]:

$$GCN(A, X) = \tilde{D}^{-1}\tilde{A}X\Theta \tag{3}$$

To solve the second issue, we propose to apply a recurrent operation to the sequence of token replays, using a Gated Recurrent Unit (GRU) as a foundation block, because they obtain similar results to an LSTM but are computationally less expensive:

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\ r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\ \tilde{h_t} &= tanh(W_h x_t + U_h(r_t \circ h_{t-1}) + b_z) \\ h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h_t} \end{aligned} \tag{4}$$

Then, the product between the learnable weights and the inputs is substituted — or the hidden state — by a graph convolution, leading to a Graph Recurrent Neural Network (GRNN) [34].

$$\begin{aligned} z_t &= \sigma(GCN(x_t, A) + GCN(h_{t-1}, A) + b_z) \\ r_t &= \sigma(GCN(x_t, A) + GCN(h_{t-1}, A) + b_r) \\ \tilde{h_t} &= tanh(GCN(x_t, A) + r_t \circ GCN(h_{t-1}, A) + b_z) \\ h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h_t} \end{aligned} \tag{5}$$

This substitution allows performing the graph convolution operation simultaneously with the recurrent operation for each timestep. In this way, the neural network can learn the spatiotemporal dependencies of the input easier than if we used a GCN first and then a GRU to take into account the time dimension (as described in subsection 4.2). Furthermore, note that we apply the GCN operation to the hidden state of the GRU so as to force this state to depend on the process model, represented, in this case, by the normalized random walk laplacian matrix. Note that the expression 5 can only deal with graphs with a fixed structure because the underlying process model never changes.

3.3 Readout

The output of the GRNN operation is the last hidden state of the GRU, which is a matrix of dimensions $\mathbb{R}^{|P|\times H}$. Since the objective of the approach is to predict the next activity of an ongoing process, we will assign a unique label to a whole graph (graph-level task). Thus, a summary operation over the graph is needed to project the output of a graph convolution operation into a single vector. This operation is called *readout*, and it reduces the graph outputs in the dimension of the nodes, i.e., the result of a readout operation is a vector $\mathbb{R}^H$. Many readout operations are available such as maximum, average, sum, or weighted average using attention. In this paper, we use the maximum operation for its simplicity and because it is suitable for classification problems [35]. Formally, let $h \in \mathbb{R}^{|P|\times H}$ be the matrix resulting from applying a graph convolution operation, then the maximum readout operation is defined as follows:

$$h_G = max(h_v) \mid v \in G \tag{6}$$

Where h_G is the summarized representation of the graph G, max is the element-wise max-pooling operation, and h_v is the learned embedding for the node v.

4 APPROACH

This section introduces our approach, "Recurrent Graph Convolutional Process Predictor" (**TACO**), which aims to address a multiclass classification problem by mapping a label to a given graph structure and its corresponding replay.

Fig. 2 illustrates the data flow of our approach. The starting point is an event log and a prefix, which serve as the basis for the prediction. Given the event log, the Split Miner is applied to generate a set of process models, each corresponding to a specific hyperparameter configuration. Subsequently, the best process model, based on its fitness on the validation set, is selected. This model is then converted into a place graph that encapsulates the relationships between the Petri net places. The graph is represented as an

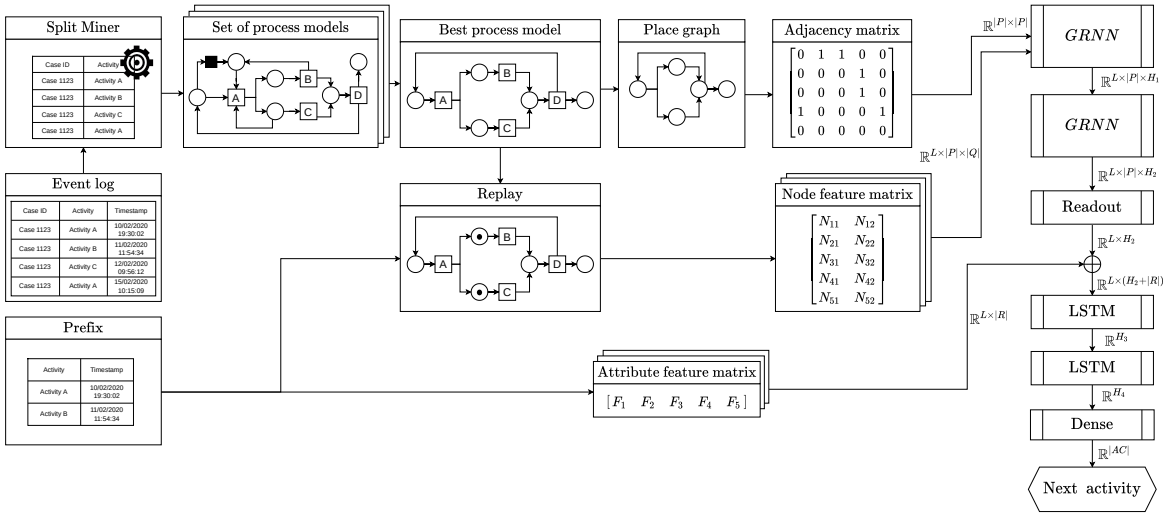

Fig. 2: Pipeline of the data flow in TACO. The sizes of the matrices are highlighted at each step.

adjacency matrix and is further normalized into a random-walk normalized Laplacian serving as one of the inputs for the Graph Recurrent Neural Network (GRNN). As the process model is common to every prefix of the log, its discovery is performed only once, just before beginning to replay any prefix.

Upon obtaining a process model, we replay the prefix on it, generating a node feature matrix that represents the activations of the places and transitions for that prefix. In addition, an attribute feature matrix containing additional information, such as the executed activities, event-related temporal data, and other attributes is created. As a result, the inputs to the GRNN consist of *(i)* the adjacency matrix and *(ii)* the node feature matrix —sequence of the activations in the Petri net after replaying the prefix—. The output of the GRNN operation, concatenated to the attribute feature matrix, is fed into an LSTM, whose outputs are subsequently passed to a softmax classifier to make the next activity prediction.

To ease the comprehension of our approach, Algorithm 1 details the steps needed to mine the best process model and obtain the process metrics that are required in order to build the feature matrices. In line 1, the best model in terms of fitness is mined using the Split Miner [36] process discovery algorithm. To do that, a grid search is performed following the same approach of [18] where the frequency threshold ϵ —which controls the filtering process— and the parallelism detection parameter η are optimized. Note that TACO works with any discovery algorithm and does not require that the process model has perfect fitness. Thus, if a prefix cannot be replayed because of a trace misalignment, the tokens will be added to the input places of the corresponding transition in order to be fired. After we obtain the best model from the Split Miner, in lines 2 to 5, the following values to build the node and attribute feature matrices are calculated: *(i)* the number of places, *(ii)* the maximum number of transitions, *(iii)* the maximum trace length, and *(iv)* the maximum number of attributes. Then, in line 6, the place graph is built, which, in turn, is represented by an adjacency matrix in line 7. In line 8, this adjacency matrix is transformed into a random walk normalized Laplacian.

Algorithm 2 builds the feature matrices used to train the neural model. The algorithm starts in line 1, where an empty node matrix is created using the dimensions related to the maximum number of transitions and the number of places. In lines 2 and 3, the lists that will contain the sequence of node matrices and the attribute matrices are created. In lines 5 to 10, we iterate over the set of events of the prefix, building the node matrix (lines 6 and 7) and the attribute matrix of each event (lines 8 and 9).

On the one hand, the function defined in line 11 builds the node matrix using an event, the best-mined process model, a previous node matrix, and the current marking of the Petri net, which is set to the initial marking in line 4 of the algorithm (when this function is called for the first time). Line 12 replays the event in the previously mined process model using the current net marking, thus obtaining the activated places, activated transitions, and the net marking after replaying the event. Then, in lines 12 to 16, the node matrix is updated with this information. Finally, in line 17, the function returns the updated node matrix and the updated marking.

Algorithm 1: Preprocessing algorithm for TACO

Input: l - whole process log, l_{train} - training set of the event log

Output: *bestModel* - best model mined, *initialMarking* - initial marking, N - number of places of the best process model, T_{max} - maximum fireable transitions in the process model, k - maximum trace length in the event log, max_{attr} - maximum number of attributes in the event log, *normalizedLaplacian* - normalized laplacian of the place graph.

1 bestModel, initialMarking = modelHyperparameterSearch(l_{train});
2 N = getNumberPlaces(bestModel);
3 T_{max} = getMaxTransitions(bestModel, l_{train});
4 k = getMaxTraceLength(l);
5 max_{attr} = maxAttributes(l);
6 placeGraph = buildPlaceGraph($bestModel$);
7 adjacencyMatrix = getAdjMatrix(placeGraph);
8 normalizedLaplacian = normalize(adjacencyMatrix);

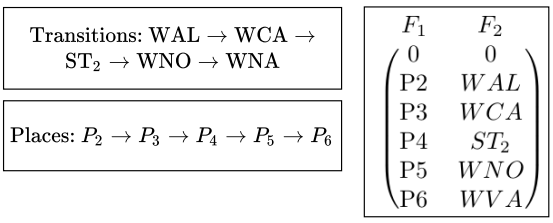

Fig. 3: Node feature matrix after encoding the prefix shown on TABLE 2 using the process model from Fig. 1.

On the other hand, the function defined in line 18 builds the attribute matrix using an event and the previously calculated maximum number of attributes. In line 19, an empty vector of the size of three plus the number of maximum attributes ($|R| = 3+m$) is created. Then, in lines 20 to 22, the time features are calculated and converted into categorical values. Finally, in lines 23 to 30, the matrix is filled with information about the activities, time, and attributes.

4.1 Encoding

In this paper, we distinguish between features specific to each node of the place graph and features inherent to each event of the prefix. The former features belong to a node feature matrix, whereas the latter ones belong to an attribute feature matrix.

A node feature matrix is defined as a matrix of dimensions $\mathbb{R}^{L\times|P|\times|Q|}$ that captures the interactions of all events of the prefix with the process model in such a way that each matrix $\mathbb{R}^{|P|\times|Q|}$ represents the interactions of an event with the process model. If the number of events in the prefix is less than L, we apply prepadding (padding on the left side of the matrix). Let Fig. 3 be an example of a node feature matrix for a single event. The first column of this matrix designated as F_1 in Fig. 3, shows which place has been activated for a given place, while each subsequent column

Algorithm 2: Algorithm for building the feature matrices of TACO

Input: outputs of Algorithm 1
Output: *attributeMatrix* - List of attribute feature matrices, *nodeMatrixList* - List of node feature matrices

```
nodeMatrix = newMatrix(N, 1 + T_max);
nodeMatrixList = [];
attributeMatrixList = [];
marking = initialMarking;
for e in p do
    nodeMatrix, marking = buildNodeMatrix(e, bestModel, nodeMatrix, marking);
    nodeMatrixList.add(nodeMatrix);
    attributeMatrix = buildAttributeMatrix(e, max_attr);
    attributeMatrixList.add(attributeMatrix);
end
Function buildNodeMatrix(e, bestModel, nodeMatrix, marking):
    act_S, act_T, marking = replayEvent(bestModel, e, marking);
    nodeMatrix[act_N, 1] = act_N;
    for i = 1; i < len(act_T); i = i + 1 do
        nodeMatrix[act_N, 1 + i] = act_T[i];
    end
    return nodeMatrix, marking;
Function buildAttributeMatrix(e, max_attr):
    attributeMatrix = newMatrix(3 + max_attr);
    timeSincePrev, timeSinceStart = getTimeFeatures(e["timestamp"]);
    qTimeSincePrev = discretizeTime(timeSincePrev);
    qTimeSinceStart = discretizeTime(timeSinceStart);
    attributeMatrix[1] = e["activity"];
    attributeMatrix[2] = qTimeSincePrev;
    attributeMatrix[3] = qTimeSinceStart;
    i = 1;
    for attribute in e do
        attributeMatrix[3 + i] = e[attribute];
        i += 1;
    end
    return attributeMatrix;
```

points out the set of transitions activated for a given event. To know which places and transitions are activated at each moment, we perform a token-replay of each event of the prefix over the process model. The features are *accumulated* in the node feature matrix, e.g., each row of the matrix in Fig. 3 represents the execution of an event. Therefore, we retain the information from the token replays of previous events of the one being processed.

Note that the number of columns of the node feature matrix depends on the number of transitions that can be fired concurrently in the Petri net, including every possible firing of silent transitions. In our example, only one transition is activated between every observable transition, so a single column, F_2, is present in the matrix. This number is

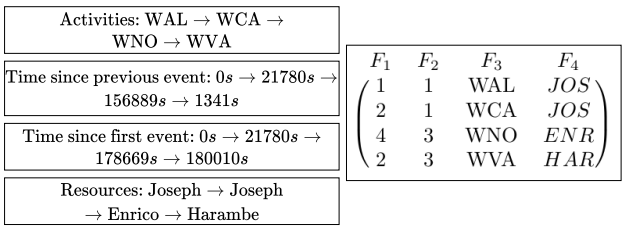

Fig. 4: Attribute feature matrix after encoding the prefix from TABLE 2.

calculated by replaying the whole log and accounting for the maximum number of possible fired transitions between two observable transitions. Note that the first column, F_1, which indicates which place has had a token after a transition firing, has the same meaning as each row of the node feature matrix, so it could be thought of as redundant. However, we still retain it since it helps the convergence of our GRNN model.

Furthermore, TACO also uses an attribute feature matrix, which encodes information about the entire prefix and has dimensions $\mathbb{R}^{L \times |R|}$. This matrix contains the following features for each event: *(i)* the event log activity whose firing generates a token in the places; *(ii)* the encoded bucket of the time since the previous event; *(iii)* the encoded bucket of the time since the first event of the prefix; and *(iv)* the m attributes associated with the event. Note that, similarly to the node feature matrix case, we apply prepadding to this matrix if necessary.

Fig. 4 shows the example of the attribute feature matrix for the trace number "214364" of the log of TABLE 2, whose model was depicted in Fig. 1. In particular, we use the time since the previous event (F_1), the time since the first event (F_2), the event log activity (F_3), and the resource (F_4). Note that all features used by our approach are categorical. To convert continuous features into categorical ones, first, we calculate every possible time difference —time between events and time since start— in the training and validation sets of the event log. Then, a quantile-based discretization is applied to this list of values, calculating equal-sized time value intervals that will be used to discretize a given value. Therefore, the interval values will span from 0 to the maximum calculated value in the training-validation event log. If a value is bigger than this latter value, it would be assigned to the highest possible interval. In the case of Fig. 4, the calculated intervals for the time since the previous event (F_1) are $[0, 1000), [1000, 20000), [20000, 150000), [150000, +\infty)$, and the calculated intervals for the time since the first event are $[0, 50000), [50000, 100000), [100000, 200000), [200000, +\infty)$. Rather than converting the categorical features to one-hot encoding, we separately embed each of the categorical features so that the dimensionality of X is independent of the number of activities of the event log and the memory consumption of the neural network is reduced. Also, note that the values for each column of the matrix are the accumulative values obtained by the token replay from the first event of the prefix to the current event. This means that the place information from previous events is not removed.

4.2 Loop overwriting

It could be thought that both the attribute feature matrix and the node feature matrix could be joined in the same matrix so that each element of the matrix, X_{ij}, represents the j feature from the place i. Thus, the GCN model could simultaneously process node and event-level information. However, this approach has the problem of overwriting the features of nodes when a loop occurs in a set of places. This is especially relevant when the loop consists of only a single activity because, in this case, the loss of information is maximum due to the low number of places involved in these types of loops. Fig. 5 exemplifies the replay process of the trace "173712" of the *BPI-2012-W-C* log from TABLE 2 using the place graph depicted also in Fig. 5. As it can be seen, the information of the feature matrix X_3 is overwritten in the feature matrix X_4, due to the presence of a loop in the activity *W_Completeren aanvraag*. Let us suppose a GCN model, which can only process the last state of a replay without fully taking into account the evolution of the trace, the only inputs would be the normalized random-walk laplacian matrix and the X_4 feature matrix. Therefore, the information available in X_3 about the resources cannot be effectively used by the GCN model.

This encoding problem could be avoided by extending the feature matrix along with the feature dimension Q up to the maximum loop size present in the event log (L) to consider older events that occurred in the same place, i.e., a feature matrix X with dimensions $\mathbb{R}^{P\times(Q\cdot L)}$. However, some event logs contain loops with a maximum length that would make this approach computationally infeasible.

A naive solution to the problem of loop overwriting would be applying the GCN operation independently to each node matrix $X_1, \cdots, X_n$ for each prefix event. Then, performing the readout operation after each convolution and feeding that information to an LSTM. However, this solution poses another problem: the structural properties of the input —the sequence of node feature matrices— and its temporal properties —the sequence of attribute feature matrices— are processed independently and, thus, some spatio-temporal interactions could be missed, such as any dependency within a loop. Therefore, in our approach, we propose to use a GRNN, as explained in Section 3.2, to simultaneously tackle the spatial and temporal features from the replay of the prefix over the process model.

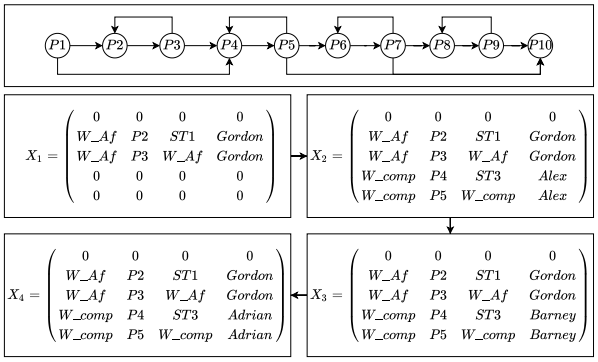

Fig. 5: Loop overwriting problem exemplified. Encoding for places P6 to P10, temporal and attribute features omitted for the sake of clarity.

5 Evaluation

5.1 Experimental setup

To perform the evaluation of TACO, we relied on the experimentation from [4] under the same conditions: a 5-fold cross-validation is performed, splitting each training fold into an 80%-20% trace distribution to obtain a validation set. The validation set is used to find the best-performing model, in terms of the epoch with the highest validation accuracy. Furthermore, we used the same attributes and compared the approaches using the "accuracy" metric, as reported in [4].

Moreover, we also used the same event logs, but "Sepsis" and "Nasa" because they cannot be applied to [15] as these logs do not contain the resources that perform the events. The characteristics of these event logs are depicted in TABLE 3, where most of them have high variability in both the temporal characteristics of the process execution —duration of events and traces— as well as the length of the traces. There are two modes of executing the approach of [18], so the two configurations are reported. Furthermore, we have extended the experimentation of [4] by adding the approaches of [16], [19] tested under the same conditions. We used the "weighted" variant of [19] due to having the most consistent results across all the datasets.

Regarding the mined process models, TABLE 4 shows the average statistics of the mined models for the cross-validation train and validation sets. The reported metrics for the process models, from left to right in the TABLE, are

Event log	Traces	Activities	Events	Avg. case length	Max. case length	Avg. event duration	Max. event duration	Avg. case duration	Max. case duration	Variants
Helpdesk	4580	14	21348	4.66	15	11.16	59.92	40.86	59.99	226
BPI-2012	13087	36	262200	20.04	175	0.45	102.85	8.62	137.22	4366
BPI-2012-C	13087	23	164506	12.57	96	0.74	30.92	8.61	91.46	4336
BPI-2012-W	9658	19	170107	17.61	156	0.7	102.85	11.69	137.22	2621
BPI-2012-W-C	9658	6	72413	7.5	74	1.75	30.92	11.4	91.04	2263
BPI-2012-O	5015	7	31244	6.23	30	3.28	69.93	17.18	89.55	168
BPI-2012-A	13087	10	60849	4.65	8	2.21	89.55	8.08	91.46	17
BPI-2013-C-P	1487	7	6660	4.48	35	51.42	2254.84	178.88	2254.85	327
BPI-2013-I	7554	13	65533	8.68	123	1.57	722.25	12.08	771.35	2278
Env-permit	1434	27	8577	5.98	25	1.09	268.97	5.41	275.84	116

TABLE 3: Statistics of the event logs used for benchmarking. Time-related measures are shown in days.

Event log	Loops	Edges	Transitions	Places	Max. rep. loop	Avg. rep. loop
BPI-2012	18404807.6	381.2	190.6	68	3	2
BPI-2012-A	5	73.6	36.8	19	0	0
BPI-2012-C	18404807.6	384.4	192.2	69.6	56	4.73
BPI-2012-O	9	48	24	13	0	0
BPI-2012-W	15.6	95.6	47.8	28.0	3	2
BPI-2012-W-C	15.6	96	48	28.2	57	4.66
BPI-2013-C-P	50	96.4	48.2	28.2	5	2.16
Helpdesk	571.6	92.8	186.4	45.6	5	2.1
BPI-2013-I	128.4	65.2	32.6	16	19	2.07
Env-permit	83187.2	255.2	127.6	52.2	3	3

TABLE 4: Average statistics of the process models.

the average number of loops, edges, transitions, places, and length and maximum length value of the loops that contain exclusively the same activity. This TABLE shows that the mined process models vary in complexity. The most simple process models are the ones mined from the event logs *BPI-2012-A* and *BPI-2012-O* since they have the lowest number of loops, transitions, and places; while the most complex event logs are the *BPI-2012*, *BPI-2012-C* and *Env-permit*. Note that the *Complete* (C) variants of the *BPI-2012* logs have more loops than their non-complete counterparts since we treat each combination of *lifecycle:transition* and activity as a separate activity.

We configured our approach with the same hyperparameters for every event log. We used 256 hidden units for both the LSTM and the GRNN and the Adam optimizer with a learning rate of 1e-3 with a cosine annealing warm restart scheduler. We also used ten buckets to quantize the time features and an embedding dimension of size 32. We trained on 100 epochs using an early stopping criterion on the validation accuracy —training is stopped if the validation accuracy does not improve during 10 epochs after reaching the maximum—. No hyperparameter tuning was performed since our approach is very resilient to a wide array of different hyperparameters. All experiments have been carried out in an Intel Xeon Gold 5220 equipped with a Tesla V100S. We implemented our approach in PyTorch 1.8.1, relying on Pm4Py [37] for processing the event logs.

We performed a two-stage statistical comparison to reduce the number of statistical tests to make and to ease the interpretation of the results. We decided to use Bayesian statistical tests instead of the classical Null Hypothesis Statistical Tests (NSHT) because they are easier to interpret and are more powerful in quantifying the differences between the approaches. First, a Bayesian approach to rank models based on the Plackett-Luce model [38] is applied. Then, a Bayesian hierarchical test [39] between our approach and the other two best approaches according to the previous ranking is performed. For this latter statistical test, the region of practical equivalence (*ROPE*) in which two approaches are statistically equal, must be defined. Following [39], a 1% accuracy value has been used as ROPE. Furthermore, in this test, we consider that one approach significantly outperforms another if the probability of obtaining the best result is greater than 95%. Note that the usefulness of the hierarchical Bayesian test resides in that the test uses the full accuracy results from the individual folds of the cross-validation testing technique, so it accounts for the fact that one approach can perform badly in one fold but very well on the rest. We rely on the R library `scmamp` to perform these statistical tests [40].

5.2 Results

TABLE 5 shows the results of the evaluation. TACO obtains the best result in 7 of the tested event logs, the second-best result in one, and the third-best in two. In particular, the highest differences with the best approach for each event log are achieved in *BPI-2013-I*, *BPI-2013-C-P*, and in *BPI-2012-W*. In the other logs, such as *BPI-2012*, *BPI-2012-C*, and *Env-permit*, the difference is less noticeable. On the other hand, TACO underperforms in the *BPI-2012-A* log, *BPI-2012-O*, and in the *BPI-2012-W-C*.

TABLE 6 shows the Bayesian Plackett-Luce model results for ranking the algorithms. Our approach obtains the best rank overall and the highest probability of being the best approach, with a difference of 24.8 percentage points over the second. Fig. 6 shows the credible intervals —5% and 95% quantiles— as well as the expected probability of winning for every tested approach. Note that a non-overlapping pair of approaches means that they are statistically different. The credible interval of the GRNN approach does not overlap any other one, which shows that our approach is statistically different in terms of ranking from the non-overlapping ones.

TABLE 7 highlights the differences between TACO and the other approaches from the state of the art, where a positive/negative difference means that TACO outperforms/underperforms the other state-of-the-art approach with which is compared to. In order to make the comparison consistent with the hierarchical statistical tests, we adopt the same ROPE strategy in which a difference of less than 1% accuracy is considered as if the approaches were equal. In general, the differences are overwhelmingly positive towards TACO. In particular, TACO outperforms the other approaches from the state of the art in 96 cases, is equal in 10 cases, and worse only in 4 cases, which correspond to the approaches of Theis (w/ attr) [18] and Hinkka [10]. On the one hand, Theis (w/ attr) outperforms TACO in *BPI-2012-W-C* by 9.64%, but TACO outperforms it in every other dataset, obtaining the largest differences in *BPI-2013-I* (26.22%) and *BPI-2012-A* (13.41%). Furthermore, Theis (w/o attr) outperforms TACO in BPI-2012-W-C by 5.86%, but fall short in every other dataset, especially in *BPI-2013-I* (18.31%) and *BPI-2012-A* (13.58%). On the other hand, our approach outperforms Hinkka [10] in 6 event logs, obtains an equal result in *BPI-2012* and *BPI-2012-C*, and underperforms [10] in *BPI-2012-A* and *BPI-2012-O*.

In general, TACO outperforms the other approaches, but in event logs with a simple process model, namely, the *BPI-2012-A* and the *BPI-2012-O*. In these cases, a recurrent network-based model is sufficient to capture the dependencies between the events. These results confirm that the graph-based approach is better suited to capture more complex behavioral patterns that are usually not identified when the learning is performed by shallower models such as LSTMs. This feature improves the prediction of the next activity when either loops or parallels are present in the log.

	BPI-2012	BPI-2012-A	BPI-2012-C	BPI-2012-O	BPI-2012-W	BPI-2012-W-C	BPI-2013-C-P	BPI-2013-I	Env-Permit	Helpdesk
Camargo	83.28	75.98	77.93	81.35	76.4	68.95	54.67	66.68	85.78	82.93
Evermann	59.33	75.82	62.38	79.42	75.37	67.53	58.83	66.78	76.19	83.66
Hinkka	**86.65**	**81.19**	**80.64**	**87.23**	84.78	70.54	**63.47**	**74.69**	84.43	83.08
Khan	42.9	74.9	47.37	66.08	60.15	52.22	43.58	51.91	83.59	79.97
Mauro	84.66	79.76	80.06	**82.74**	85.98	68.64	24.94	36.67	53.59	31.79
Pasquadibisceglie	83.25	74.12	74.6	78.88	81.19	68.34	47.45	46.03	**86.69**	83.93
Tax	**85.46**	79.53	**80.38**	82.29	85.35	69.79	**64.01**	**70.09**	85.71	**84.19**
Theis (w/o attributes)	82.89	65.5	75.26	78.38	**86.22**	**80.06**	59.48	59.41	**86.29**	78.77
Theis (w/ attributes)	80.96	65.67	75.75	76.89	**86.86**	**83.84**	54.65	51.5	85.12	79.69
Venugopal	54.69	54.88	63.75	67.87	53.9	64.8	48.44	49.62	69.58	78.7
Zararah	85.31	**79.87**	77.43	81.15	85.19	68.19	63.29	68.94	86.06	**83.96**
TACO	**87.08**	**79.78**	**80.85**	**82.95**	**88.34**	**74.2**	**67.53**	**77.72**	**87.66**	**85.2**

TABLE 5: Mean accuracy of the 5-fold cross-validation. Best, second-best, and third-best approaches are highlighted in cyan, orange and yellow, respectively.

	TACO	Tax	Hinkka	Zararah	Camargo	Theis (w/o)	Theis (w/)	Pasqua.	Evermann	Mauro	Venugopal	Khan
Rank	1.03	2.59	2.83	3.92	5.07	6.81	7.31	7.62	8.27	10.02	11.14	11.4
Prob. (%)	39.7	14.9	13.6	9.2	6.3	3.8	3.4	3.1	2.6	1.6	1	0.9

TABLE 6: Plackett-Luce rankings and posterior probabilities of the approaches.

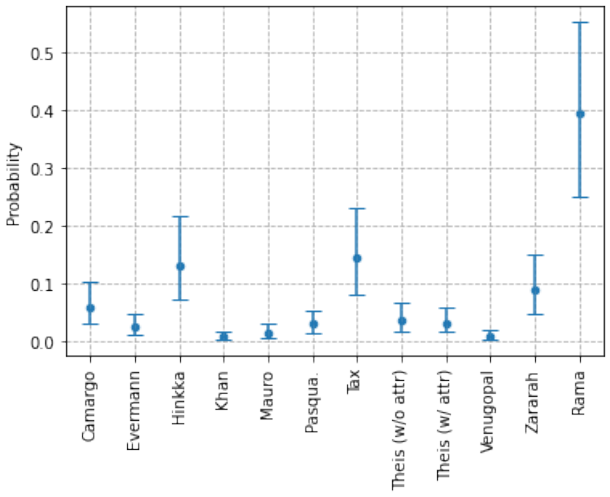

Fig. 6: Credible intervals (5%, 95% quantiles) and the expected probability of winning for the tested approaches.

To further assess the differences between the approaches, we applied a hierarchical Bayesian test whose results are shown in TABLE 8. This table, given two approaches, A and B, shows the probability of A being better than B ($A > B$), the probability of the two approaches being equal ($A = B$), and the probability of A being worse than B ($A < B$). Furthermore, we can assess whether A is not worse than B by summing the probabilities from $A > B$ and $A = B$. In this test, it is assumed that a probability greater than 95% means statistical significance. We perform this test to compare our approach against the two best ones according to the Plackett-Luce ranking, as we did in the analysis of the accuracy differences. These tables report the probabilities for each dataset and the overall probability of the approach is better.

On the one hand, the statistical tests show that TACO is significantly better in 5 datasets —underlined in TABLE 8—, equal in two of them —*BPI-2012* and *BPI-2012-C*—, worse in other two —*BPI-2012-A* and *BPI-2012-O*—, and not significantly better in *BPI-2013-C-P* These results are aligned with those shown in TABLE 7 except for *BPI-2013-C-P*, in which the statistical analysis does not give significance due to the high variability in the results of the cross-validation. In that case, it can only be affirmed that TACO is not worse than Hinkka [10]. Overall, even though TACO is not statistically better than Hinkka [10], it still obtains a probability of 91% of being better in terms of accuracy. On the other hand, the statistical tests also show that TACO is significantly better than Tax [14] in 5 datasets —underlined in TABLE 8—, equal in the *BPI-2012-C* and we cannot affirm anything in the rest. Regarding these latter datasets, TACO is very close in *Env. permit* (short of a 2%), while in the other three, we can only affirm that TACO is not worse than [14]. Despite this, TACO still has a probability of being better than [14] of 99.3%, which is significant.

5.3 Ablation study

In this section, the results of two different ablation studies are presented. In the first one, we use every piece of information possible —activities, token-replay, time features, and event attributes— to test the different components individually, so as to assess how they influence the performance of the whole approach. In the second one, we leave the architecture of TACO unchanged to test the influence of different pieces of information used by our approach. To reduce the number of experimentations and facilitate the interpretation of the results, this second ablation study is focused on evaluating the performance of the approach when using *(i)* only transition activations and the place marking, and *(ii)* only time features and event attributes. We do not test the usage of only activities as it would make the usage of the GRNN module of TACO pointless.

The goal of the first ablation study is to test *(i)* how the GRNN and the LSTM work individually and *(ii)* how a GRNN would operate without recurrence, i.e., how the GCN operator would perform, when it only can process the last accumulated replay from the whole prefix. In this case, we aim to test how much performance we gain by

	BPI-2012	BPI-2012-A	BPI-2012-C	BPI-2012-O	BPI-2012-W	BPI-2012-W-C	BPI-2013-C-P	BPI-2013-I	Env-permit	Helpdesk
Camargo	3,8	3,1	2,92	1,6	11,94	5,25	12,86	11,04	1,88	2,27
Evermann	27,75	3,26	18,47	3,53	12,97	6,67	8,7	10,94	11,47	1,54
Hinkka	0,43	-2,11	0,21	-4,28	3,56	3,66	4,06	3,03	3,23	2,12
Khan	44,18	4,18	33,48	16,87	28,19	21,98	23,95	25,81	4,07	5,23
Mauro	2,42	-0,68	0,79	0,21	2,36	5,56	42,59	41,05	34,07	53,41
Pasqua.	3,83	4,96	6,25	4,07	7,15	5,86	20,08	31,69	0,97	1,27
Tax	1,62	-0,45	0,47	0,66	2,99	4,41	3,52	7,63	1,95	1,01
Theis (w/o attr)	4,19	13,58	5,59	4,57	2,12	-5,86	8,05	18,31	1,37	6,43
Theis (w/ attr)	6,12	13,41	5,1	6,06	1,48	-9,64	12,88	26,22	2,54	5,51
Venugopal	32,39	24,2	17,1	15,08	34,44	9,4	19,09	28,1	18,08	6,5
Zarahah	1,77	-0,79	3,42	1,8	3,15	6,01	4,24	8,78	1,6	1,24

TABLE 7: Accuracy cross-validation differences of TACO against other approaches.

	TACO vs. Hinkka			TACO vs Tax		
	$A > H$	$A = H$	$A < H$	$A > T$	$A = T$	$A < T$
BPI-2012	0.305%	99.68%	0.01%	98.02%	1.97%	0.0136%
BPI-2012-A	0.028%	2.168%	97.80%	24.45%	72.85%	2.70%
BPI-2012-C	1.627%	99.82%	0.02%	2.34%	97.62%	0.036%
BPI-2012-O	0.072%	0.185%	99.74%	51.99%	44.55%	3.45%
BPI-2012-W	100%	0%	0%	99.98%	0.019%	0%
BPI-2012-W-C	99.88%	0%	0.111%	99.62%	0.358%	0.026%
BPI-2013-C-P	81.16%	14.54%	4.30%	99.87%	0.12%	0.012%
BPI-2013-I	99.29%	0.658%	0.053%	99.85%	0.137%	0.017%
Env-permit	99.02%	0.898%	0.08%	93.03%	6.8%	0.166%
Helpdesk	99.55%	0.439%	0%	54.58%	45.39%	0.031%
Overall	91%	1.2%	7.8%	99.3%	0.2%	0.6%

TABLE 8: Hierarchical bayesian tests comparing TACO (A) against Hinkka et al. (H) [10] and Tax et al. (T) [14].

avoiding the problem of loop overwriting described in Section 4.2. The results of this ablation study are shown in TABLE 9, where the GCN performs comparably worse w.r.t. the GRNN in every log, especially in logs characterized by a high process model complexity, which hinders the performance of the GCN —results for these logs are underlined in TABLE 9.

Furthermore, every event log has a fair number of loops, which further degrades the performance of the GCN. Comparing the LSTM with TACO, the greater differences are reached in the logs highlighted in bold where, the LSTM is unable to capture the interactions between the events, so the usage of structural information from the process model is most beneficial. However, note that the LSTM outperforms the GRNN in almost every log because the GRNN is restricted to processing the information from the token replay, while the LSTM has access to the information about activities, time features, and attributes.

	GCN	GRNN	LSTM	TACO
BPI-2012	53.43	82.83	86.71	87.08
BPI-2012-A	79.49	79.49	79.77	79.78
BPI-2012-C	75.12	77.26	80.31	80.85
BPI-2012-O	72.53	80.82	82.64	82.95
BPI-2012-W	31.35	65.17	88.06	88.34
BPI 2012 W C.	67.29	68.07	73.75	74.2
BPI-2013-C-P	57.56	59.57	65.29	**67.53**
Helpdesk	83.47	83.75	79.99	**85.2**
BPI-2013-I	60.44	67.15	75.84	**77.72**
Env-permit	84.25	84.60	85.90	**87.66**

TABLE 9: Ablation study comparing the individual performance (accuracy) of the different components of TACO.

The results of the second ablation study are shown in TABLE 10, showing how the different pieces of information influence the performance when compared with the results of our original approach, TACO. As shown in this TABLE, the difference between using only the information of the activated transitions, the information of the marking places, or both, is negligible in these datasets. However, we cannot assure that using only the node marking or the transition activation would guarantee this behavior in every possible event log, so TACO adds both pieces of information in order to be as general as possible. Moreover, the use of time features is beneficial in most cases —underlined in the aforementioned table—, achieving the best results in *BPI-2012-O* and in *BPI-2012-C*. Finally, the use of event attributes is also positive in most cases —most significant improvements are highlighted in bold in TABLE 10—. Most notably, the *BPI-2012-A* does not show any improvement from using additional information, which is an example of a simple log in which the token replay information is enough to obtain the best achievable result with this architecture.

5.4 Loop behavior

TABLE 11 shows some detailed statistics about the presence of loops in the event logs, as well as the performance of TACO when compared to the LSTM in the ablation study (already described in TABLE 9). Furthermore, TABLE 12 shows a fine-grained list of the presence of loops in the event

	ANT	AN	AT	ANTI	ANTIR
BPI-2012	85.42	85.40	85.42	85.70	87.08
BPI-2012-A	79.49	79.49	79.49	79.78	79.78
BPI-2012-C	77.61	77.59	77.57	80.21	80.85
BPI-2012-O	81.14	81.18	81.16	83.00	82.95
BPI-2012-W	85.36	85.31	85.36	86.14	88.34
BPI-2012-W-C	68.46	68.49	68.56	68.91	**74.2**
BPI-2013-C-P	61.88	62.49	62.76	63.42	**67.53**
Helpdesk	84.08	83.96	83.99	84.20	85.20
BPI-2013-I	69.03	68.68	68.62	70.22	**77.72**
Env-permit	86.24	86.21	86.28	86.00	87.66

TABLE 10: Ablation study showing the performance (accuracy) of our approach using different inputs, represented by a different initial, being "A" activities, "N" activated places, "T" activated transitions, "I" time features, and "R" log attributes.

	Total cases	Avg. cases length	Cases with loops	% cases with loops	Number of loop appearances	LSTM	TACO	Difference
BPI-2012	13087	20.04	5643	43.12	9342	86.71	87.08	0.37
BPI-2012-C	13087	12.57	5712	43.64	9932	80.32	80.85	0.53
BPI-2012-A	13087	4.65	0	0	0	79.77	79.78	0.01
BPI-2012-O	5015	6.23	350	6.98	356	82.65	82.95	0.3
BPI-2012-W	9658	17.61	6583	68.16	10810	88.06	88.34	0.28
BPI-2012-W-C	9568	7.5	7011	72.59	13075	73.76	74.2	0.44
BPI-2013-C-P	1487	4.48	449	30.2	566	65.29	67.53	2.24
BPI-2013-I	7554	8.68	7114	94.18	10970	75.84	77.72	1.88
Env-permit	1434	5.98	30	2.09	32	85.9	87.66	1.76
Helpdesk	4580	4.66	950	20.74	1021	79.99	85.2	5.21

TABLE 11: Loop statistics of the tested process logs.

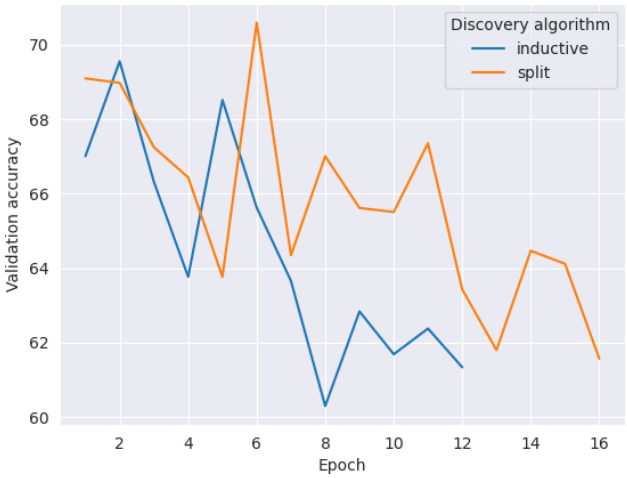

Fig. 7a: Validation accuracy in *BPI-2013-C-P*

logs. For each event log, two columns are presented: the first one, "Len.", refers to the number of different activities that compose a loop, whereas the second one shows the number of loop instances with that length in the log. For example, two loops such as "$AAAAA$" and "$AAAA$" would be accounted as a tuple "$(A, 2)$" with $Len. = 1$ and $Num. = 2$. In these logs, most loops have few activities but with high incidence, like *BPI-2012-W*, whose average case and loop length are 17.61 and 2, respectively.

As shown by TABLE 11, the best results are achieved in the highlighted logs, which have a high number of loops and a short average case length, with the exception of *Env-permit*. This shows that our approach is able to extract better the loop information when the traces have a high number of loops. Furthermore, when no loops are available in the event log, such as in *BPI-2012-A*, the performance of the LSTM and TACO is virtually the same.

In conclusion, the performance of TACO depends on two main factors: the complexity of the underlying process model and the information provided by the timestamps and event attributes of the event log. On the one hand, as shown in TABLE 11, TACO provides an edge when the underlying process model is fairly complex. In these cases, an LSTM is unable to capture every complex relationship between the events of the log. On the other hand, the usage of the information from the timestamps and event attributes proves to be useful in most cases, as shown in TABLE 10. However, in some logs, such as *BPI-2012-A*, the underlying process model is too simple —our predictive model captures the same information as any model not based on structural information—, and the usage of the information from the timestamps and event attributes is not useful —as shown by the results of TABLE 10—; so, in that cases, the performance will not be improved by TACO w.r.t an LSTM that only uses the information from the activities of the event log.

5.5 Process discovery algorithm effect

In this section, we conducted a series of experiments to evaluate the sensitivity of TACO to the process discovery algorithm used for obtaining the underlying process model that describes the observed behavior in the event log. To this end, we compared the performance of our approach using both Split Miner and Inductive Miner [41]. The Inductive Miner is a widely adopted algorithm that aims to discover sound and fitting models in a finite time. Based on the divide-and-conquer principle, this algorithm recursively partitions the event log to create a process tree, which is subsequently converted into a Petri net. On the other hand, the Split Miner aims to discover a BPMN model, which is, in turn, converted into a Petri net. To do so, it begins by obtaining a directly follows graph (DFG) of the event log, which is subsequently pruned and refined by discovering XOR and AND gateways. We opted to use the Inductive Miner for the comparison due to its popularity and effectiveness in process discovery tasks and because it employs a different approach than the Split Miner, so the discovered process models by this algorithm have different structural properties.

In our experiments, both algorithms were configured to discover models with perfect fitness, ensuring a fair comparison. The results of this experimentation are highlighted in TABLE 13, where it can be seen that there are very small differences in the results obtained by TACO with both algorithms. The biggest differences are in the event logs *BPI-2013-C-P* and *BPI-2012-W-C*, with a variation in performance of 0.9% and 0.34%, respectively. These results confirm that TACO is insensitive to the algorithm used to discover the process model.

Figures 7a and 7b show how the validation accuracy evolves using both model miners in the second cross-validation fold for *BPI-2013-C-P* and *BPI-2012-W-C*. For *BPI-2013-C-P*, TACO obtains an accuracy of 65.99% and 68.31% using Inductive Miner and Split Miner, respectively; while for *BPI-2012-W-C*, it gets 74.09% and 74.92%, respectively. These results correspond with the highest point in the validation accuracy plots of Figures 7a and 7b —epoch 6 for both event logs—. On the one hand, TACO with Split Miner takes slightly longer to trigger the early stopping criterion, an effect that is evidenced by the fact that it takes more epochs to stop training. On the other hand, TACO with Split Miner achieves higher validation accuracies during training, since in both figures the maximum accuracy is always reached with the Split Miner algorithm.

BPI-2012	Len.	1	2	3	4	6	7	8	9	10	11	12	13	14	15	16	17	18	23	25
	Num.	153	9018	1	1	4	74	14	25	7	14	11	5	6	3	3	2	2	1	1
BPI-2012-C	Len.	1	2	3	4	5	6	7	8	9	10	11	13	14	19					
	Num.	9719	8	26	18	90	32	18	8	4	3	1	2	2	1					
BPI-2012-W	Len.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	18	19	21	23
	Num.	160	10378	121	1	45	4	30	17	7	11	4	13	2	6	5	3	1	1	1
BPI-2012-W-C	Len.	1	2	3	4	5	6	7	8	9	10	11	13	14	16	19				
	Num.	12962	11	31	22	24	9	5	4	1	1	1	1	1	1	1				
BPI-2012-O	Len.	4	5	8																
	Num	320	30	6																
BPI-2013-C-P	Len.	1	2	3	4															
	Num.	388	163	13	2															
BPI-2013-I	Len.	1	2	3	4	5	6	7	9	11										
	Num.	7897	2737	173	111	23	15	11	2	1										
Helpdesk	Len.	1	2	3	4															
	Num.	855	149	12	5															
Env-permit	Len.	1	2	3																
	Len.	3	28	1																

TABLE 12: Fine grained loop statistics.

	TACO S.M.	TACO I.M.	Difference
BPI-2012	87.00	86.83	0.17
BPI-2012-A	79.92	79.79	0.13
BPI-2012-C	80.89	80.90	-0.01
BPI-2012-O	83.07	82.96	0.11
BPI-2012-W	88.29	88.32	-0.03
BPI-2012-W-C	74.53	74.19	0.34
BPI-2013-C-P	67.86	66.96	0.9
BPI-2013-I	77.58	77.37	0.21
Env-Permit	87.65	87.56	0.09
Helpdesk	85.10	85.15	-0.05

TABLE 13: Performance of TACO (accuracy) using the Split Miner (S.M.) and the Inductive Miner (I.M.).

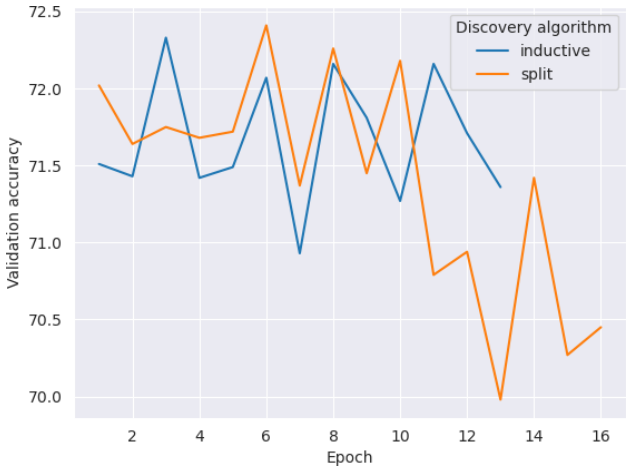

Fig. 7b: Validation accuracy in *BPI-2012-W-C*

5.6 Hyperparameter effect

In this section, a search of hyperparameters was conducted to evaluate their influence on the performance of our approach. Considering that TACO has 12 different hyperparameters, each with three possible values and that 10 event logs are used in a 5-fold cross-validation to evaluate the influence of the hyperparameters in the performance of TACO, we would have had to train $3^{12} \cdot 5 \cdot 10 = 26,572,050$ different models. Thus, our experimentation has been restricted to only four hyperparameters that directly affect the graph-based part of our approach: the hidden size of the first and second GRNN layers —"Hidden 1" and "Hidden 2", respectively— and the dropout of the first and second GRNN layers –"Dropout 1" and "Dropout 2", respectively— .

Furthermore, in order to reduce the number of models tested, we only tried three different values for each hyperparameter, giving 81 possible combinations. In particular, the values of 64, 128, 256, and 0, 0.1, 0.2 were used for each possible hidden size and value of dropout [41], respectively. Finally, we performed a grid search on the first fold of two representative event logs: *Helpdesk* and *BPI-2012-W-C*.

Fig. 8a, 8b, 9a, and 9b exhibit the parallel coordinate plots for each hyperparameter evaluated in the two event logs. To simplify the interpretation of the figures, we separated the coordinate plots for each dataset into two distinct figures, encompassing the 10 best-performing sets of hyperparameters and the 10 worst-performing sets of hyperparameters, respectively, for each evaluated event log[1]. Thus, Fig. 8a and 8b depict the parallel coordinate plots for each hyperparameter combination on the Helpdesk event log. The first plot, Fig. 8a, showcases the 10 best-performing evaluations while Fig. 8b, portrays the 10 worst-performing ones. As evidenced by these figures, most of the combinations successfully reach the optimal maximum with roughly 84% accuracy, barring a few notable exceptions with validation accuracies ranging from 18% to 71%. In this case, the failure to attain a satisfactory validation accuracy is directly linked to using a dropout value exceeding 0 in the second layer of the GRNN, whereas the first layer can maintain a dropout value above 0 without necessarily failing to converge. Fig. 9a and 9b present the same coordinate plots, but for the BPI-2012-W-C event log. In this situation, most combinations reach a local optimum of approximately 72%, aside from those that fall short due to a dropout value higher than 0 in the second layer of the GRNN.

6 Conclusions and Future Work

In this paper, we presented TACO, an approach that, contrary to most approaches, solves the next activity prediction problem by using both the information readily available in the traces of the event log and the information available in the process model. Thus, we proposed a novel architecture that simultaneously leverages the structural information

1. It is available the full results of this experimentation in the supplementary material.

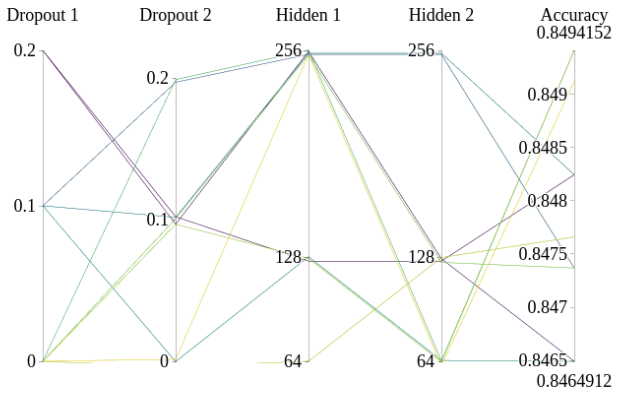

Fig. 8a: Parallel coordinate plot of the 10 best-performing hyperparameters in the Helpdesk event log.

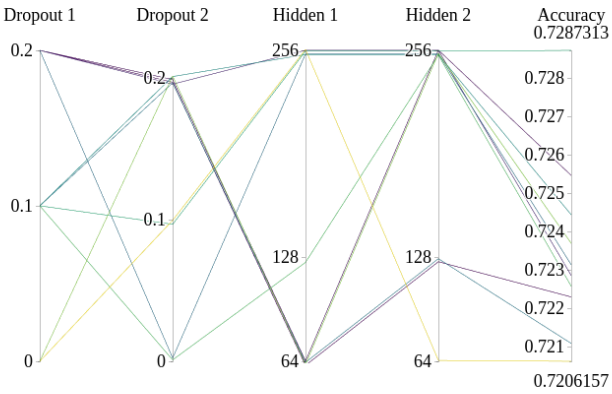

Fig. 9a: Parallel coordinate plot of the 10 best-performing hyperparameters in the BPI-2012-W-C event log.

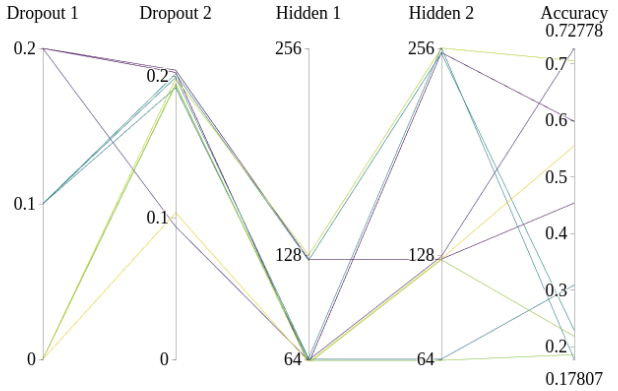

Fig. 8b: Parallel coordinate plot of the 10 worst-performing hyperparameters in the Helpdesk event log.

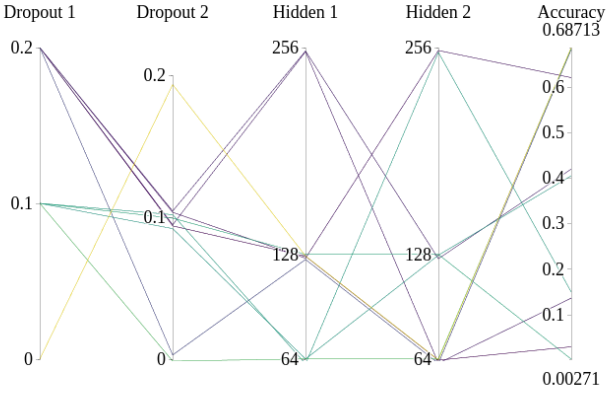

Fig. 9b: Parallel coordinate plot of the 10 worst-performing hyperparameters in the BPI-2012-W-C event log.

from the process model and the information available in the events of the event log. For that, we modified the definition of a GRU to allow the simultaneous processing of the state of the Petri net using GCNs for each event of the prefix. Thus, our architecture can learn from the prefixes of the event log in two dimensions at the same time: spatial, represented as a directed graph of places of a Petri net, and temporal represented as the sequence of events of the prefix. Furthermore, contrary to GCNs, our encoding scheme does not lose information when a loop occurs in the process and can capture the dynamics of the replay of the events in the process model while being memory efficient and easy to compute.

We evaluated our proposal in 10 real-life event logs and compared it against 10 approaches from the state-of-the-art. The results show that TACO works more consistently and obtains better results overall than the other approaches. Our experimentation shows the adequacy of using GNNs for predictive process monitoring since their internal structure can better leverage the information of the process model. This property is also clearly verified in the results, as TACO outperforms the other approaches in every event log except those with a simpler process model.

For future work, we intend to test more different graph convolution operators, extend the approach to other process models, such as resource models, and tackle more predictive monitoring tasks such as the remaining time prediction.

7 Acknowledgments

This work has received financial support from the Consellería de Educación, Universidade e Formación Profesional (accreditation 2019-2022 ED431G-2019/04), the European Regional Development Fund (ERDF), which acknowledges the CiTIUS - Centro Singular de Investigación en Tecnoloxías Intelixentes da Universidade de Santiago de Compostela as a Research Center of the Galician University System, and the Spanish Ministry of Science and Innovation (grants PDC2021-121072-C21 and PID2020-112623GB-I00). E. Rama-Maneiro is supported by the Spanish Ministry of Education, under the FPU national plan (FPU18/05687).

Furthermore, the authors also wish to thank the supercomputer facilities provided by CESGA.

8 Biographies

EFRÉN RAMA-MANEIRO received the B.Eng. degree in computer engineering and the M.Sc. degree in Big Data from the University of Santiago de Compostela, Spain in 2018 and 2019 respectively. He is a Researcher and currently working toward the Ph.D. degree at the Centro Singular de Investigación en Tecnoloxías Intelixentes, University of Santiago de Compostela. His research interests include process mining and deep learning.

JUAN C. VIDAL received the B.Eng. degree in computer science from the University of La Coruña, La Coruña, Spain, in 2000, and the Ph.D. degree in artificial intelligence from the University of Santiago de Compostela (USC), Santiago de Compostela, Spain, in 2010, where he was an Assistant Professor with the Department of Electronics and Computer Science, from 2010 to 2017. He is currently an Associate Researcher with the Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), USC. His research interests include process mining, fuzzy logic, machine learning, and linguistic summarization.

MANUEL LAMA received the Ph.D. degree in physics from the University of Santiago de Compostela, in 2000, where he is currently an Associate Professor of artificial intelligence. He has collaborated on more than 30 projects and research contracts financed by public calls, participating as a principal investigator in 20 of them. These activities were implemented in areas, such as process discovery, predictive monitoring, and management of dynamic processes. As a result of this research, he has published over 150 scientific articles with review process in conference and national and international journals.

References

[1] W. M. P. van der Aalst et al., "Process mining manifesto," in *Proceedings of the 9th International Business Process Management Workshops (BPM 2011)*, ser. Lecture Notes in Business Information Processing, vol. 99. Springer, 2011, pp. 169–194.

[2] F. M. Maggi, C. D. Francescomarino, M. Dumas, and C. Ghidini, "Predictive monitoring of business processes," in *Advanced Information Systems Engineering*. Springer International Publishing, 2014, pp. 457–472.

[3] N. Tax, I. Teinemaa, and S. J. van Zelst, "An interdisciplinary comparison of sequence modeling methods for next-element prediction," *Software and Systems Modeling*, 2020.

[4] E. Rama-Maneiro, J. Vidal, and M. Lama, "Deep learning for predictive business process monitoring: Review and benchmark," *IEEE Transactions on Services Computing*, pp. 1–1, 2022.

[5] N. Mehdiyev, J. Evermann, and P. Fettke, "A multi-stage deep learning approach for business process event prediction," in *Proceedings of the 2017 IEEE 19th Conference on Business Informatics (CBI 2017)*. IEEE, 2017, pp. 119–128.

[6] ——, "A novel business process prediction model using a deep learning method," *Business & Information Systems Engineering*, 2018.

[7] F. Taymouri and et al., "Predictive business process monitoring via generative adversarial nets: The case of next event prediction," in *Lecture Notes in Computer Science*. Springer International Publishing, 2020, pp. 237–256.

[8] N. D. Mauro, A. Appice, and T. M. A. Basile, "Activity prediction of business process instances with inception CNN models," in *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 348–361.

[9] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba, "Using convolutional neural networks for predictive process analytics," in *2019 International Conference on Process Mining (ICPM)*. IEEE, jun 2019.

[10] M. Hinkka, T. Lehto, and K. Heljanko, "Exploiting event log event attributes in RNN based prediction," in *Communications in Computer and Information Science*. Springer International Publishing, 2019, pp. 405–416.

[11] A. Khan, H. Le, K. Do, T. Tran, A. Ghose, H. Dam, and R. Sindhgatta, "Memory-augmented neural networks for predictive process analytics," Feb. 2018.

[12] K. Heinrich, P. Zschech, C. Janiesch, and M. Bonin, "Process data properties matter: Introducing gated convolutional neural networks (GCNN) and key-value-predict attention networks (KVP) for next event prediction with deep learning," vol. 143, p. 113494, apr 2021.

[13] J. Evermann, J.-R. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *Decision Support Systems*, vol. 100, pp. 129–140, aug 2017.

[14] N. Tax, I. Verenich, M. L. Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in *Advanced Information Systems Engineering*. Springer International Publishing, 2017, pp. 477–492.

[15] M. Camargo, M. Dumas, and O. González-Rojas, "Learning accurate LSTM models of business processes," in *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 286–302.

[16] Z. A. Bukhsh, A. Saeed, and R. M. Dijkman, "Processtransformer: Predictive business process monitoring with transformer network," 2021.

[17] H. T. C. Nguyen, S. Lee, J. Kim, J. Ko, and M. Comuzzi, "Autoencoders for improving quality of process event logs," *Expert Systems with Applications*, vol. 131, pp. 132–147, oct 2019.

[18] J. Theis and H. Darabi, "Decay replay mining to predict next process events," *IEEE Access*, vol. 7, pp. 119 787–119 803, 2019.

[19] I. Venugopal, J. Tollich, M. Fairbank, and A. Scherp, "A comparison of deep-learning methods for analysing and predicting business processes." IEEE, jul 2021.

[20] S. Weinzierl, "Exploring gated graph sequence neural networks for predicting next process activities," in *5th International Workshop on Artificial Intelligence for Business Process Management (AI4BPM2021)*, 07 2021.

[21] W. M. van der Aalst, "A practitioner's guide to process mining: Limitations of the directly-follows graph," *Procedia Computer Science*, vol. 164, pp. 321–328, 2019, cENTERIS 2019 - International Conference on ENTERprise Information Systems.

[22] A. Graves and et al., "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, oct 2016.
[23] A. Jalayer, M. Kahani, A. Beheshti, A. Pourmasoumi, and H. R. Motahari-Nezhad, "Attention mechanism in predictive business process monitoring." IEEE, oct 2020.
[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015.
[25] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 933–941.
[26] M. Daniluk, T. Rocktäschel, J. Welbl, and S. Riedel, "Frustratingly short attention spans in neural language modeling," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.
[27] I. J. Goodfellow and et al., "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 2014, pp. 2672–2680.
[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 30th Annual Conference on Neural Information Processing Systems (NIPS 2017)*, 2017, pp. 5998–6008.
[29] B. van Dongen, "Bpi challenge 2012," 2012.
[30] M. Kirchmer, *High Performance Through Business Process Management*. Springer International Publishing, 2017.
[31] J. Desel and W. Reisig, "Place/transition petri nets," in *Lectures on Petri Nets I: Basic Models*. Springer Berlin Heidelberg, 1998, pp. 122–173.
[32] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017*, 2017.
[33] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *The Semantic Web*. Springer International Publishing, 2018, pp. 593–607.
[34] L. Ruiz, F. Gama, and A. Ribeiro, "Gated graph recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 6303–6318, 2020.
[35] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.
[36] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, and A. Polyvyanyy, "Split miner: automated discovery of accurate and simple business process models from event logs," *Knowledge and Information Systems*, vol. 59, no. 2, pp. 251–284, may 2018.
[37] A. Berti, S. J. van Zelst, and W. M. P. van der Aalst, "Process mining for python (pm4py): Bridging the gap between process- and data science," 2019.
[38] B. Calvo, J. Ceberio, and J. A. Lozano, "Bayesian inference for algorithm ranking analysis," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, jul 2018.
[39] A. Benavoli, G. Corani, J. Demsar, and M. Zaffalon, "Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis," *J. Mach. Learn. Res.*, vol. 18, pp. 77:1–77:36, 2017.
[40] B. Calvo and G. Santafé, "scmamp: Statistical Comparison of Multiple Algorithms in Multiple Problems," *The R Journal*, vol. 8, no. 1, pp. 248–256, 2016.
[41] N. Srivastava and et al., "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.