Encoding High-Level Control-Flow Construct Information for Process Outcome Prediction

Citation for published version (APA):

DOI:
10.1109/ICPM57379.2022.9980737

Document status and date:
Published: 14/12/2022

Document Version:
Typeset version in publisher’s lay-out, without final page, issue and volume numbers

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.
Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaaccess@tue.nl
providing details and we will investigate your claim.
Encoding High-Level Control-Flow Construct Information for Process Outcome Prediction

Mozhgan Vazifehdoostirani  
Eindhoven University of Technology  
The Netherlands  
m.vazifehdoostirani@tue.nl

Laura Genga  
Eindhoven University of Technology  
The Netherlands  
l.genga@tue.nl

Remco Dijkman  
Eindhoven University of Technology  
The Netherlands  
r.m.dijkman@tue.nl

Abstract—Outcome-oriented predictive process monitoring aims at classifying a running process execution according to a given set of categorical outcomes, leveraging data on past process executions. Most previous studies employ Recurrent Neural Networks to encode the sequence of events, without taking the structure of the process into account. However, process executions typically involve complex control-flow constructs, like parallelism and loops. Different executions of these constructs can be recorded as different event sequences in the event log. This makes it challenging for a recurrent classifier to detect potential relations between a high-level control-flow construct and the prediction target. This is especially true in the presence of high variability in process executions and lack of data. In this paper, we propose a novel approach which encodes the control-flow construct each event belongs to. First, we exploit Local Process Model mining techniques to extract frequently occurring control-flow constructs, instead of mining local process models representing the most precisely, we propose to extract process structural information on control-flow for outcome-oriented PPM. More frequently occurring control-flow constructs, instead of mining start-to-end process models. As a result, control-flow constructs which may have a relation with the outcome of the process may be missing or only partially represented in start-to-end process models. As a result, control-flow constructs which may have a relation with the outcome of the process may be missing or only partially represented in start-to-end process models.

In this work, we investigate an alternative way to exploit structural information available in the process models such as loop and concurrency can be beneficial for some PPM tasks [13]. However, these methods mainly rely on start-to-end process models that are either available or mined from the event log to extract process structure information. From literature it is well-known that, due to the high variability of processes in many real-world scenarios, deriving a complete process model often leads to either a too simplified model representing only a portion of the process behaviors, or to so-called spaghetti-like models, which often allow for (almost) any sequence of events captured in the event log [15], [16]. As a result, control-flow constructs which may have a relation with the outcome of the process may be missing or only partially represented in start-to-end process models.

In this work, we investigate an alternative way to exploit information on control-flow for outcome-oriented PPM. More precisely, we propose to extract process structural information by applying Local Process Model (LPM) discovery [16] to extract portions of process models representing the most frequently occurring control-flow constructs, instead of mining start-to-end process models.

To the best of our knowledge, only one previous study, presented in [17], has investigated the benefits of exploiting...
information from LPMs in predicting patients’ outcome of palliative treatments. However, their approach encodes such information by a basic one-hot encoding, and only tested it with classical machine learning methods. We extend to extend the previous study to the LSTM network, as this is the most commonly used DL method in PPM [10], and introduce novel encoding methods that can be used in LSTM classifiers. Accordingly, the present work addresses the following research questions:

**RQ1** How can we encode control-flow information into suitable input data for an LSTM classifier?

**RQ2** Can we improve the performance of outcome prediction using control-flow information in an LSTM model?

The remainder of this paper is organized as follows. Section II reviews the relevant related work. Section III provides basic concepts used throughout the paper. Section IV introduces the proposed methodology. Section V discusses the experimental setup and results. Finally, Section VI draws some conclusions and delineates some ideas for future studies.

## II. RELATED WORK

Previous research implemented different deep learning architectures, including LSTM [18], [14], Convolutional Neural Network (CNN) [4], [5], and Graph Neural Network (GNN) [19] achieving a varying degree of accuracy and performance. Despite the fact that they have employed diverse architectures, most of them primarily relied on the sequence of events and data payload corresponding to each event to predict how a running case will evolve in the future. Some studies encoded activities using one-hot vector encoding, then enriching it with additional numerical data related to each event, such as execution time [9], [18], [20]. These approaches faced dimension challenges when dealing with lengthy prefixes. A different set of studies employed embedding instead of one-hot encoding and showed a noticeable improvement in the performance of the LSTM [21], [22], and CNN [4] classifiers. Considering resource information [21] and experience [23] have also been studied in previous research and showed benefits in some prediction tasks and event logs.

Few recent studies have investigated the opportunity to include information on the structural aspect of the process in the PPM models to boost prediction performance. Given the fact that graph encoding is a convenient way of representing process executions [24], some of them suggested to encode the process structure information into graph-based models, then using GNN for the prediction. The approach proposed by Venugopal et al. [25] extracts Directly-Follows Graphs for building a model of the process, then using it with a Graph Convolutional Neural Network (GCNN) to learn the prediction. Chiorrini et al. [19] implemented a Gated Graph Neural Network to predict the next activity by building Instance Graphs from the process model. Another set of works in the literature have proposed to encode structural information from the process model for RNN models. For example, Metzger and Neubauer used encoding methods for representing information on the parallel branch each activity belongs to (derived by the overall process model) and encode this information for training LSTM models [13]. Di FrancescoMarino et al. [26] proposed to pre-process log traces to derive information on loops, then using this information, possibly together with domain knowledge related to execution constraints, to improve the performance of the next-activity prediction. Despite the fact that these methods often achieve good performance, many of them focus only on a subset of the possible control-flow behaviors, or require a well-defined start-to-end process model. However, high process variability makes extracting a single start-to-end process model challenging [15].

In addressing the high variability of processes, recent studies focused on alternative approaches to applying process discovery techniques to provide evidence-based insights into the structural relation between events and frequent patterns. Some studies employed abstraction methods such as Fuzzy Miner [27] and Heuristic Miner [28], which try to abstract uncommon and noisy behaviors in order to represent the core process behaviors. In addition, clustering techniques [29] may be used to construct a group of models from more homogeneous behaviors. A different strategy to deal with variable processes is extracting only a subset of process behaviors that are of interest according to some user-defined criteria. For this purpose, we intend to derive LPMs representing the common control-flow constructs (e.g., choice, concurrency, loops) by learning the most frequent behavioral patterns from a given event log. Experiments conducted on real-world datasets have shown that this approach is able to learn insightful patterns that would not be properly captured in start-to-end models mined by process discovery algorithms [16].

## III. PRELIMINARIES

This section introduces important concepts used throughout the paper. The input of most process mining methods is an event log \( \mathcal{L} \) composed of traces, where each trace records a sequence of events, each conveying information about the executed activities on single process executions (also referred to as cases) [2].

**Definition 1 (Event):** Let \( \mathcal{A} \) be the universe of activities, \( \mathcal{C} \) be the universe of case identifiers, \( \mathcal{T} \) be the time domain, and \( \mathcal{D}_i \) be the set of additional attributes with \( i \in [1, m] \), \( m \in \mathbb{Z} \). An event is a tuple of \( e = (a, c, t, d_1, \ldots, d_m) \), where \( a \in \mathcal{A} \), \( c \in \mathcal{C} \), \( t \in \mathcal{T} \), and \( d_i \in \mathcal{D}_i \).

**Definition 2 (Trace, Prefix Trace, Event log):** A trace \( \sigma_n, n = 1, 2, \ldots, N \), represents the \( n^{th} \) execution of a process consisting of a finite non-empty sequence of \( \ell_n = [\sigma_n] \) discrete events in which their timestamp does not decrease. Let \( \mathcal{E} \) be the set of events; we define the following mapping functions to map each event to its corresponding timestamp by \( \pi_T : \mathcal{E} \rightarrow \mathcal{T} \), to its case identifier by \( \pi_C : \mathcal{E} \rightarrow \mathcal{C} \), and to its activity by \( \pi_A : \mathcal{E} \rightarrow \mathcal{A} \). For each trace \( \sigma_n = (e_1, e_2, \ldots, e_{|\sigma_n|}) \) we must have \( \pi_C(e_i) = \pi_C(e_j), \forall i, j \in [1, |\sigma_n|] \), and \( \pi_T(e_i) \leq \pi_T(e_j), \text{if } i < j \). An event log \( \mathcal{L} \) is a set of traces. We define a Prefix Trace, \( P_{\sigma_n}^k = (e_1, e_2, \ldots, e_k) \), of arbitrary length \( 1 \leq k \leq |\sigma_n| \) as a trace sub-sequence that begins at the beginning of the trace \( \sigma_n \).
**Definition 3 (Prefix Trace Encoder):** Let $X$ be the set of prefix traces. A Prefix Trace Encoder is a function $g : X \rightarrow W$ that takes as input a prefix trace and returns a feature vector in $W$.

![Fig. 1. An example process tree and its equivalent Petri-net [16]](image)

**Definition 4 (Outcome Classifier):** Let $\Phi$ be a set of nominal values representing possible classification outcomes. An Outcome Classifier is defined as a function $f : W \rightarrow \Phi$, that takes an encoded vector of a trace prefix and returns the corresponding outcome. For example, an outcome value might represent the satisfaction of a constraint on the cycle time of the case, or the validity of a temporal logic constraint.

**Definition 5 (Local Process Model):** Let $\Omega = \{\rightarrow, \times, \land, \lor\}$ be the set of operators indicating sequence, choice, concurrency, and loop behavior in a process, respectively. A Local Process Model $LN$ is a (process) tree, where each leaf node is an activity $a \in A$ and each non-leaf node an operator $\omega \in \Omega$. LPMs are often represented as Petri nets [16]. An example of an LPM in the form of a process tree and its conversion into a Petri net is shown in Fig. 1.

**Definition 6 (LPM Discovery Function):** A LPM Discovery Function gets an event log $\mathcal{L}$ and returns the set $\Theta = \{LN_1^2, \ldots, LN_k^2\}$ of all LPMs satisfying user-defined thresholds over five quality criteria [16]: support (i.e., the number of occurrences of the LPM), confidence (i.e., the fraction of events of the activities in $\mathcal{L}$ which fit the behavior described by the LPM), language fit (which measures the portion of behaviors allowed by the LPM that is observed in $\mathcal{L}$), determinism (which relates to the degree to which future behavior can be determined), and coverage (which measures the frequency of the activities described by the LPM in $\mathcal{L}$).

IV. METHODOLOGY

The overview of the proposed method is depicted in Fig 2. Given an event log $\mathcal{L}$, we aim to build a classifier to predict the outcome of a running case considering the encoded control-flow construct information derived from LPMs. To this end, we first mine the set of LPMs. Then, we introduce an LPM feature generator function and two prefix trace encoding methods to encode the prefix traces enriched with the LPM feature into suitable input data for LSTM models. The encoded prefixes are then given as input to the classifier.

**A. LPM Feature Generation**

In this study, we follow the discovery method proposed by Tax et al. [16] to extract the LPMs from the event log. After discovering the set of LPMs, the trace prefix set is created, by generating for each trace $\sigma_n$ all prefix traces $P\sigma^n_k$ for $2 \leq k \leq |\sigma_n|$. Then, conformance checking techniques [30] are used to check whether one or more LPMs occur in each prefix trace. More precisely, we introduce the LPM feature $LA$ as a new event attribute denoting the set of LPMs each event in a prefix trace belongs to. To avoid uncertainty due to partial occurrences of LPMs in defining the LPMs feature, we only consider complete occurrences of LPMs. Namely, an event is marked as belonging to one LPM only if all the activities belonging to the LPM occur in the appropriate order in the corresponding prefix trace. Note that it can happen that an LPM involves one or more activities for which multiple events of a trace can be matched. In that case, our approach will select for each activity the event that comes first in the trace. Different matching strategies can be explored like, for instance, matching the events in such a way to minimize the time in between the activities of the LPM instance. We plan to explore more sophisticated matching strategies in future work.

An example of prefix traces enriched with the LPMs feature is shown in Fig. 3. For the sake of simplicity, we only show the activity and the $LA$ feature for each event. For example, the prefix trace $\langle A, \{LN_2^2\}, (B, \emptyset), (C, \{LN_2^2\}) \rangle$ contains three events with activity classes $A, B, C$ respectively, where the first and the third event belong to the LPM $LN_2^2$.

![Example Prefix Trace](image)

**B. Encoding Prefix Traces**

We introduce two novel encoding methods to encode the prefix traces enriched with the LPM feature. These techniques are based on the two most common event encoding approaches in previous PPM studies for LSTM models, i.e., one-hot encoding and embedding layers [6], and differ in the amount of encoded information (and, consequently, in the complexity of the encoding). The following subsections delve into the characteristics of each encoding.

1) Wrapped-One-Hot encoding: The one-hot encoding is used to encode categorical values in a binary representa-
In this context, each event \( e \) is encoded as a binary vector \( \mathcal{V} \) of length \( |\mathcal{A}| \). The encoding defines an arbitrary but consistent index to each activity over set \( \mathcal{A} \) where \( \text{index} : \mathcal{A} \rightarrow \{1, 2, \ldots, |\mathcal{A}|\} \). Then, it assigns value 1 to feature number \( \text{index}(\pi_A(e)) \), and value 0 to the rest of the elements of the vector. We refer to this form of encoding as \( OH_{\text{Act}} \) from now on. A straightforward way to add information about LPMs in this encoding would be to add the one-hot encoded vector of LPMs to each one-hot encoded event \( e \). However, a possible drawback of this solution is dimension explosion, since using one-hot encoding each additional feature can result in many features to add to the one-hot vector of the process activities. To tackle this challenge, we wrap the defined LPMs feature into \( OH_{\text{Act}} \) using a scalar transformation, instead of extending the one-hot vector with these patterns. By doing so, we keep the same dimension of the original vector of activities. From now on, we call this encoding \( \text{Wrapped-One-Hot encoding} \) (WOH).

More precisely, we multiply vector \( \mathcal{V} \) with a scalar \( s \) to create vector \( \mathcal{V}' \), which wraps the LPMs feature. We use two different approaches for defining the scalar \( s \). Let \( \pi_L : \mathcal{E} \rightarrow 2^{\Theta_L} \) be a mapping function that maps each event to the LPMs it belongs to. The first approach, called \( \text{Wrapped-One-Hot-Br} \) (WOHB), utilizes a function \( I(\pi_L(e)) = 1 \), if \( \pi_L(e) \neq \emptyset \) and \( I(\pi_L(e)) = 0 \) otherwise. As a result, we can define \( s = I(\pi_L(e)) + 1 \).

The second encoding strategy, named \( \text{Wrapped-One-Hot-Fr} \) (WOHF), aims to encode the frequency of LPMs corresponding to each event. In this case, \( s = |\pi_L(e)| + 1 \).

The Wrapped one-hot encoding provides a simple means to identify events belonging to at least one LPM (or to how many, depending on the encoding used), thus distinguishing them from those events for which no LPMs could be detected. However, when several LPMs can occur in the same execution, this representation is likely to provide little information, since many or most of the activities in the prefix will have the same value in the encoded vector. Furthermore, since it does not represent information related to which LPMs an event belongs to, it offers little support in recognising high-level control-flow constructs from the sequential prefix trace. To deal with these limitations, we use the embedding layers.

2) Embedding Layers: Embedded encoding comes mostly from NLP and the information retrieval domain to create highly informative but low-dimensional vectors [31]. Inspired by previous PPM studies using embedding layers to take into account categorical attributes [14], [32], we exploit embedding layers to represent the relation between events and their corresponding process behaviors (i.e., the LPMs) while limiting the feature vector size.

We define two embedding layers, one for the set of activities and one for the set of LPMs. We define an arbitrary but consistent function \( ID : \mathcal{A} \rightarrow \{1, 2, \ldots, |\mathcal{A}|\} \) as done in previous work for embedding layers [32]. Thus, if we have a prefix trace \( P^0 e_n = \langle e_1, e_2, \ldots, e_k \rangle \) we transform it to \( P^0 e_n = \langle ID(\pi_A(e_1)), ID(\pi_A(e_2)), \ldots, ID(\pi_A(e_k)) \rangle \). In this way, we have a unique integer value in place of each activity inside the trace.

The transformed prefix trace is given as input to an embedding layer in the neural network, which starts with generating a vector of a given size with random numbers for each activity. During the training of the network, the random values are gradually adjusted via backpropagation; namely, the embedding layer is optimized to learn similarities between activities so as to minimize the loss function for the specific outcome prediction problem. The embedded vector dimension is determined during the hyperparameter tuning. We call this Embedding Activity Embedding (Emb_{Act}) from now on.

In a similar way we define LPMs Embedding (Emb_{LPMs}) approach by encoding the sequence of corresponding patterns to each event in a prefix trace using the defined LPMs feature. To this end, we define an arbitrary but consistent function \( ID_L : \Theta_L \rightarrow \{1, 2, \ldots, 2^{\Theta_L}\} \). Then, given the prefix trace \( P^0 e_n = \langle e_1, e_2, \ldots, e_k \rangle \) we transform it to \( P^0 e_n = \langle ID(\pi_L(e_1)), ID(\pi_L(e_2)), \ldots, ID(\pi_L(e_k)) \rangle \). This sequence is then given as input to the embedding layer. Besides embedding the sequence of activities and corresponding LPMs, we also concatenate the two embedded vectors for LSTM training to combine activity and LPMs features named \( \text{Emb}_{\text{Act+LPMs}} \) (Emb_{Act+LPMs}) encoding.

Fig. 4 zooms in the training part of the schema in Fig. 2 for the \( \text{Emb}_{\text{Act+LPMs}} \) encoding, showing a schematic of the generation of the encoded vector. Two embedded activities and LPMs vectors are concatenated after concatenating layer in such a way to create one vector for each events consisting of its (embedded) activity (shown in gray) and LPMs (shown in white) before feeding to the LSTM layers.
V. EXPERIMENTAL SETUP

This section discusses a set of experiments carried out to evaluate the proposed approach. In the previous section, we discussed two different encoding approaches we introduced to answer RQ1: How can we encode control-flow information for an LSTM classifier? Here, we intend to study the performance of the proposed encoding strategies to answer RQ2: Can we improve the performance of outcome prediction using control-flow information in an LSTM model?

The following subsections describe the experimental settings, the datasets and the experimental results.

A. Implementation

For mining the LPMs set, we used the LPM miner plugin available in ProM 6.9\(^1\) based on [16] and the default plugin settings for extracting LPMs from each event log. We implemented our approach for the LPM feature generator, prefix trace encoding, and LSTM models with Python\(^2\).

B. Settings

We implemented an LSTM architecture with two different input layers based on the two proposed encoding methods. One LSTM is built with a normal input layer for \(OH_{Act}\), \(WOH_B\), and \(WOH_F\) encoding methods. Another LSTM contains embedding (and concatenating) layers for the \(Emb_{Act}\), \(Emb_{Act+LPMs}\), and \(Emb_{LPMs}\) encoding strategies.

We implemented a single LSTM network to predict all prefix sizes and zero-padding is used to ensure the same size of the input data. To reduce the dimension of input data and the complexity of the models, we trim traces after the point that 90% of cases with the minority outcome class has finished with an upper bound of 40 events per trace, as suggested in [1]. We have also filtered out incomplete traces for which there is no outcome recorded.

We first divide each event log into testing and training sets and then perform the LPM discovery method only on training cases. More precisely, we divided each dataset into 80% for training and 20% for final testing. We shuffled cases and randomly divided cases into train and test while keeping the distribution of case length and outcome classes in the test set the same as the training set. In particular, as we split event logs on the level of cases, different prefix traces of the same case remain either in the training or the testing set. Note that, in doing this, we hold an assumption that there is no concept drift in the business process, and past structural behaviors in a business process can be used to predict future outcomes.

For the hyperparameter optimization, we used the implementation of the Tree-structured Parzen Estimator (TPE) [33] in Python. The optimization phase is conducted using 10% of the training set as a validation set. The range of possible values for parameters explored in this phase is reported in Table I. To avoid over-fitting, we also performed early stopping by halting training when loss on the validation set does not improve for 10 consecutive epochs, and we added dropout to each LSTM layer. The dropout rate is defined by hyperparameter tuning.

C. Datasets

To evaluate the proposed method, we have used public event logs widely used in the literature, accessible from the 4TU Centre for Research Data\(^3\). We have considered the datasets used in [1] with same labeling strategy for the current research.

Production represents a manufacturing process with a limited number of cases but high variants. The outcome of each trace is defined based on whether the number of rejected work orders exceeds zero or not.

BPIC2012 contains the execution history of a loan application process in a Dutch financial institution. We define three binary outcomes (hence, three datasets) for cases based on whether their loan application is accepted (BPIC2012-ac), rejected (BPIC2012-re), or canceled (BPIC2012-ca).

BPIC2011 assembles patients’ medical history from the Gynaecology department of a Dutch Academic Hospital. Similar to previous works, we defined four datasets with binary outcomes for cases from this event log referring to [1] based on the satisfaction of different temporal constraints on the order of occurrence of tasks in a case, named BPIC2011-f1, BPIC2011-f2, BPIC2011-f3, BPIC2011-f4.

\(^1\)https://www.promtools.org/doku.php

\(^2\)https://github.com/MozhganVD/LPMforPPM

\(^3\)https://data.4tu.nl/
too high loss of information. Indeed, neglecting information related to the activities, leads to a reasonable, because when we use only LPMs features we miss information about activities that are not inside any discovered LPMs; as a result, we may miss discriminative activities.

Comparing the reported AUC in Table II, encoding the combination of activity sequences and LPMs feature into LSTM models leads to a consistent improvement of at least one performance measure (but all of them in most cases) for all event logs, except for Traffic Fine.

We performed the Friedman test to statistically test whether the improvement in the performance of models including LPMs feature is significant [35]. We focus on the AUC metric since, as mentioned above, it has the benefit of being independent from the threshold. The Friedman test is a non-parametric test which ranks the methods for each data set separately with a null hypothesis that there is no significant difference in their ranking. Considering the AUC of each evaluated method for ranking methods in the Friedman test, the null hypothesis is rejected with \( p\text{-value} \leq 0.05 \). In order to further assess the relative performance of each pair of methods, we used the Nemenyi test for two groups of encoding separately. In this way, we can assess whether adding the LPMs feature could increase performance without taking into account the effect of different encoding on performance improvement. In particular, the critical difference diagrams in Fig 5 show that using LPMs features leads to higher prediction performance in both encoding group methods using a 0.05 significance level.

### E. Discussion

The results show a consistent improvement in the prediction performance due to the use of LPMs, thus showing that encoding control-flow construct information has a positive impact on the performance of case outcome prediction. As expected, the encoding with embedding layers usually outperforms their one-hot encoding counterpart.

At the same time, however, the performance improvements are quite little for some of the tested datasets, with improvements around (or less than) 1% or 2%, while in the Traffic Fine we observed a (slight) reduction (around 0.04%). To shed some light on the reason for the obtained performance, we need to consider the characteristics of the event logs and of the discovered LPMs. Indeed, we expect LPMs involving non sequential behaviors to be the most beneficial in terms of classification performance, since they offer the higher abstraction means. To delve into this aspect, we have computed for each LPMs set of each dataset the average percentage of different common control-flow constructs (i.e., sequence, choice, parallel, direct loop, and indirect loop) covered by the set of extracted LPMs, shown in Table III. For each activity inside each LPM, we count the number of branches which are in parallel or alternative to the activity branch, together with the number of short or indirect loops it belongs to. If there were none of these structures, we assumed it was involved only in a sequence. We then went through all LPMs and count these values for each activity (cumulative). At the end,

### Table II: Evaluation Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Encoding Group</th>
<th>Encoding</th>
<th>AUC</th>
<th>Accuracy</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPIC2011</td>
<td>One-hot encoding</td>
<td>OH</td>
<td>64.18%</td>
<td>76.34%</td>
<td>58.50%</td>
</tr>
<tr>
<td></td>
<td>Embedding</td>
<td>Emb</td>
<td>67.78%</td>
<td>73.75%</td>
<td>62.54%</td>
</tr>
<tr>
<td>BPIC2011</td>
<td>One-hot encoding</td>
<td>OH</td>
<td>61.87%</td>
<td>54.85%</td>
<td>51.54%</td>
</tr>
<tr>
<td></td>
<td>Embedding</td>
<td>Emb</td>
<td>64.2%</td>
<td>84.01%</td>
<td>77.67%</td>
</tr>
<tr>
<td>BPIC2012</td>
<td>One-hot encoding</td>
<td>OH</td>
<td>81.98%</td>
<td>80.95%</td>
<td>80.10%</td>
</tr>
<tr>
<td></td>
<td>Embedding</td>
<td>Emb</td>
<td>82.12%</td>
<td>81.29%</td>
<td>80.54%</td>
</tr>
</tbody>
</table>

**Traffic Fine** corresponds to a road traffic management system. The labeling is established based on whether the fine was paid in whole or if it was referred to credit collection.

### D. Results

We measured the performance of each model w.r.t common measurements, including area under the ROC curve (AUC), accuracy, and weighted F1-score. Among mentioned metrics, AUC is the more reliable measure to compare different methods because it is a threshold-independent measure and not biased in the case of highly imbalanced data [34]. We considered a 0.5 threshold value for the other measurements. Table II shows the average performance of each classifier for the tested event logs. Bold numbers show the best performance for each dataset regarding two main encoding scenarios.

A first observation that can be drawn from Table II is that, in general, encoding only information related to LPMs, neglecting information related to the activities, leads to a too high loss of information. Indeed, \( Emb_{A|LPM} \) encoding performed generally poorly in all the evaluated scenarios, with few exceptions (e.g., the production dataset). This result is reasonable, because when we use only LPMs features we miss information about activities that are not inside any discovered LPMs; as a result, we may miss discriminative activities.
Fig. 5. Pairwise comparison of LSTM models with (a) embedding, and (b) variations of One-hot encoding methods using Friedman-Nemenyi test.

TABLE III
OVERVIEW OF THE EVENT LOGS AND DISCOVERED LPMs CHARACTERISTICS

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Metrics</th>
<th>BPIC11-f1</th>
<th>BPIC11-f2</th>
<th>BPIC11-f3</th>
<th>BPIC11-f4</th>
<th>BPIC12-ac</th>
<th>BPIC12-re</th>
<th>BPIC12-ca</th>
<th>Production</th>
<th>TrafficFine</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPMs</td>
<td>sequence</td>
<td>75.3%</td>
<td>78.8%</td>
<td>58.7%</td>
<td>81.5%</td>
<td>72.4%</td>
<td>78.7%</td>
<td>67.1%</td>
<td>74.6%</td>
<td>91.3%</td>
</tr>
<tr>
<td>choice</td>
<td>2.6%</td>
<td>0.0%</td>
<td>23.1%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.3%</td>
<td>7.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>parallel</td>
<td>7.6%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>1.5%</td>
<td>0.6%</td>
<td>1.4%</td>
<td>1.4%</td>
<td>8.7%</td>
<td></td>
</tr>
<tr>
<td>Direct loop</td>
<td>14.0%</td>
<td>21.2%</td>
<td>18.2%</td>
<td>16.4%</td>
<td>23.1%</td>
<td>17.3%</td>
<td>23.1%</td>
<td>15.2%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>Indirect loop</td>
<td>0.5%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>2.1%</td>
<td>3.0%</td>
<td>3.1%</td>
<td>1.4%</td>
<td>8.7%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>Overlapped LPMs</td>
<td>80.0%</td>
<td>75.0%</td>
<td>84.0%</td>
<td>86.1%</td>
<td>88.6%</td>
<td>95.5%</td>
<td>85.7%</td>
<td>95.1%</td>
<td>70.0%</td>
<td></td>
</tr>
<tr>
<td>Covered activities</td>
<td>9%</td>
<td>4%</td>
<td>0%</td>
<td>15%</td>
<td>44%</td>
<td>63.9%</td>
<td>42%</td>
<td>27%</td>
<td>60%</td>
<td></td>
</tr>
<tr>
<td>Event logs</td>
<td>Max. trace length</td>
<td>36</td>
<td>32</td>
<td>30</td>
<td>30</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>23</td>
<td>10</td>
</tr>
<tr>
<td>Med. trace length</td>
<td>25</td>
<td>54</td>
<td>21</td>
<td>44</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>Pos. class ratio</td>
<td>49%</td>
<td>78%</td>
<td>23%</td>
<td>28%</td>
<td>48%</td>
<td>17%</td>
<td>32%</td>
<td>53%</td>
<td>46%</td>
<td></td>
</tr>
<tr>
<td>Variant</td>
<td>815</td>
<td>977</td>
<td>793</td>
<td>977</td>
<td>3578</td>
<td>3578</td>
<td>3578</td>
<td>3578</td>
<td>203</td>
<td>185</td>
</tr>
</tbody>
</table>

for each control-flow construct we have a vector of values, each corresponding to one activity, from which we compute the average values for each control-flow constructs. We have reported the fraction of each control-flow constructs to all observed behaviors (in percentage) to assess which one is mostly supported by extracted LPMs set.

We have also calculated the percentage of overlapped LPMs discovered from each event log (Overlapped LPMs) and the percentage of activity classes covered by all discovered LPMs (Covered activities). Additional information about the event logs such as, e.g., the median and maximum (after truncating lengthy traces) are reported in Table III.

According to the table, most of the mined LPMs show a prevalence of sequential behaviors, which can partly justify the limited improvement in performance. Indeed, in this context, the LPMs can still provide useful abstractions when they occur in log traces where different activities occur in between the activities modeled by an LPM (recall that LPMs show eventually follow relations); however, when the behavior is completely sequential, not much useful information is added. The second most frequent construct is the direct loop, with a percentage of 14% or more in all the datasets; on the contrary, parallel and choice constructs are mostly very infrequent (with the exception of the choice construct in BPIC11-f3, where it achieves 23%). Overall, these results do not allow to detect a relation between the different constructs and the magnitude of the increase of the classification performance.

It is worth noting that the highest amount of sequential behaviors is shown by the Traffic Fine dataset; indeed, despite the fact that this dataset shows a relatively high amount of parallel behaviors with respect to the other datasets (8.7%), the large majority of behaviors captured by LPMs are sequential (91.3%). This observation suggests that discovered LPMs do not boost the LSTM model with new information about non-sequential behaviors. Another unique characteristics of Traffic Fine dataset is having short traces with median length of four events per trace. This implies that most prefix traces are actually too short to show complete occurrence of LPMs; consequently, the added LPMs features is a sparse matrix adding noise instead of helpful information.

Overall, these results suggest that an interesting direction to explore to improve the classification performance is to tailor the extraction of LPMs to the classification task. In this study, we used the default setting for the LPM extraction; however, these settings have been designed to fulfill a process discovery task, i.e., to generate patterns showing a good balance between support, precision and generalization. However, such criteria may not be the best choice for a classification task. For instance, an LPM showing a lower precision but involving a higher amount of parallelism may be more beneficial for classification purposes than sequential patterns. Similarly, taking into account potential correlations between the mined patterns and the outcome class could lead to extract patterns which may be otherwise filtered out because of a low support. On the same line, taking into account domain knowledge in determining patterns expected to have an impact on the outcome can also be a valuable means to support the classification task. Furthermore, we did not take into account other attributes corresponding to each event in this study; however, in some datasets there could be an interesting relation between extracted LPMs and other event attributes which could be used to boost prediction results. We plan to explore these directions in future work.

VI. CONCLUSION AND FUTURE WORK

This paper examined the impact of encoding high-level control-flow construct information on the process outcome
prediction problem. We employed LPM discovery to derive frequent control-flow constructs which may not be observed through a start-to-end process model. Then, we studied two novel encoding methods to encode prefix traces including corresponding frequent control-flow constructs for LSTM classifiers. The experimental results have shown that using the proposed method we are able to improve the performance of outcome prediction tasks consistently. Additionally, the results showed that embedding layers mostly outperform the one-hot encoding technique. The results also suggest that encoding control-flow construct information is expected to lead to better results in datasets whose process executions are long enough to allow the detection of interesting patterns. By delving into the structure of the tested datasets we also observed that the mined LPMs capture mostly sequential behaviors, which can justify the limited improvements in the performance of the classifier in several datasets.

In future work, we intend to design an integrated methodology to extract LPMs based on the characteristics of each event log and their impact on the outcome prediction task. We also plan to investigate the impact of different event matching strategies for generating the LPM feature. Additionally, we are interested to design a suitable encoding method for graph-based neural networks, to exploit their ability to encode process structures to encode the specific relations occurring within the detected LPMs.

REFERENCES


