PROMISE: Coupling Predictive Process Mining to Process Discovery

Vincenzo Pasquadibisceglie[a,*], Annalisa Appice[a,b], Giovanna Castellano[a,b], Wil van der Aalst[c]

[a]*Department of Informatics, Università degli Studi di Bari Aldo Moro, via Orabona, 4 - 70125 Bari - Italy*
[b]*Consorzio Interuniversitario Nazionale per l'Informatica - CINI, Italy*
Tel.: +39-080-5443262
[c]*RWTH Aachen University, Aachen, Germany*

Abstract

Process discovery, one of the main branches of process mining, aims to discover a process model that accurately describes the underlying process captured within the event data recorded in an event log. In general, process discovery algorithms aim to return models describing the entire event log. However, this strategy may lead to discover complex, incomprehensible process models concealing the correct and/or relevant behavior of the underlying process. Processing the entire event log is no longer feasible when dealing with large amounts of events. In this study, we propose the PROMISE$^+$ method that rests on an abstraction involving predictive process mining to generate an event log summary. This summarization step may enable the discovery of simpler process models with higher precision. Experiments with several benchmark event logs and various process discovery algorithms show the effectiveness of the proposed method.

Keywords: Process discovery, Predictive process mining, Deep learning, Abstraction, Event log summarization

*Corresponding author

Email addresses: `vincenzo.pasquadibisceglie@uniba.it` (Vincenzo Pasquadibisceglie), `annalisa.appice@uniba.it` (Annalisa Appice), `giovanna.castellano@uniba.it` (Giovanna Castellano), `wvdaalst@pads.rwth-aachen.de` (Wil van der Aalst)

1. Introduction

Process-Aware Information Systems (PAISs) are software systems that are being used increasingly by public and private organizations to manage and execute operational processes involving people, applications and/or information sources. Examples of PAISs are workflow management systems, case-handling systems and enterprise information systems. In principle, these systems are driven by process models [17] that can be represented in a graphical language, e.g., in a Petri-net-like notation. In practice, most PAISs do not use an explicit process model, i.e., there are implicit, emerging processes based on best practices analysis. This is because processes are commonly invisible and often exist as abstract concepts, hence they are frequently difficult to materialize [18]. In addition, even when processes are documented in some way, they are often described using a variety of notations [18]. Fortunately, more and more detailed information about the execution of process instances (e.g., activities being executed) are being recorded in the form of event logs [19]. These event logs represent the key enabler for process discovery algorithms [18].

Process discovery is a branch of process mining [19] that is concerned with the automatic discovery of process models from event logs. Existing process discovery algorithms are formulated in process mining [19] and strike different trade-offs between the accuracy in capturing the behavior recorded in an event log and the complexity of the derived process model. Ensuring the quality of discovered process models is one of the main issues of the current research in process discovery. In fact, to be useful the process model should: (1) parse the traces in the log, (2) parse traces that are not in the log but are likely to belong to the process that produced the log and (3) not parse other traces. The first property is called *fitness*, the second one *generalization*, and the third one *precision*. On the other hand, the process model should be as simple as possible. The simplicity property is usually quantified via complexity measures [2].

Most of the process discovery algorithms tend to use the whole event log for discovery. This makes no longer effective traditional process discovery algorithms in the emerging big data settings, where the event data may be too big to be entirely processed with standard hardware in a reasonable time [51]. A straightforward solution to overcome this problem is to build summaries of huge event logs by down-sizing the amount of event data to be processed with a process discovery algorithm. To this purpose, various filtering [8, 38, 39, 44] and sampling [40, 37, 16] methods have been proposed as preprocessing step for process discovery. These methods apply extractive approaches to identify important traces (sampling) or relevant excerpts from traces (filtering) in event logs, and reproduce them verbatim as part of the log summary.

In this study, we explore how an abstraction-based approach can be effectively used as an alternative to the extraction strategy performed by filtering or sampling. In particular, the goal is to use an abstraction strategy to distill the most important information from an event log, in order to produce a synthesized version of the log useful for the process discovery task. This idea is borrowed from the text summarization field [15], where abstraction-based methods are proved to be commonly more effective than extraction-based methods for realizing text summaries. When applied to event logs, traditional filtering and sampling methods extract a subset of existing traces (or excerpts of exiting traces) for inclusion in the summary output. Opposed to these extraction-based approaches, we propose an abstraction-based method that learns a generative representation (abstraction) of an event log that can be helpful to capture important process information possibly spread across traces. This abstraction is used for generating few "new" representative traces that comprise more flexible activity sequences and provide a compact representation of the initial event log content. These new traces are subsequently processed with process discovery algorithms to better achieve the ideal balance between the quality and the complexity of the process models finally discovered.

This idea goes in some way into the direction of [36], where clustering is used to group traces, while cluster medoids are selected as representative, existing traces to populate the log summary. However, differently from [36], our method creates a log summary by creating new representative traces, apart from extract-ing existing significant ones. In particular, the key idea of our method, named PROMISE$^+$ (coupling Predictive pROcess MIningto ProceSs DiscovEry), is to learn the generative abstraction of the event log by setting a predictive process mining method.

Predictive process mining is a recently emerged family of process mining methods to predict the unfolding of running traces (e.g., the next activity) based on the knowledge learned from a historical event log. With the recent boom of deep learning, several accurate predictive process mining methods leveraging deep neural networks have been proposed [47, 46, 7, 32, 35, 33, 34]. Following this research stream, we use a deep neural network with Long Short-Term Memory (LSTM) layers [22] to process executed activities by taking into account the sequential nature of events recorded in event logs. The trained LSTM model is used within a trace generation technique to produce new representative traces by iteratively sampling from the network's output distribution, then feeding in the sample as input at the next step. Finally, we wrap a process discovery algorithm in a forward trace selection strategy that selects the summary traces that contribute to maximizing the quality of the process model discovered.

The paper is organized as follows. Section 2 overviews recent advances of literature in process discovery and outlines the motivation of this study. Section 3 reports preliminary concepts, while Section 4 describes the proposed PROMISE$^+$ method. In Section 5, we present an extensive empirical study that compares PROMISE$^+$ to sampling and filtering baselines taken from the recent literature on process mining. Finally, Section 6 draws conclusions and sketches the future work.

2. Background and Motivation

Several process discovery algorithms (e.g., Alpha Miner [48], Inductive Miner [25, 26], Heuristic Miner [50], Fodina [49] and Split Miner [2]) have been proposed in the recent process mining literature. Although Alpha Miner [48] and the basic Inductive Miner [25] have been originally designed to depict as much as possible behaviors seen in the event log into the process model, several process discovery algorithms use filtering. For example, Hybrid ILP Miner [53], Heuristic Miner [50], Fodina [49] and Split Miner [2] have been designed to filter infrequent behaviors within their internal data structure, in advance to discovering a process model. Also, the Inductive Miner family has introduced filtering mechanisms to filter infrequent directly-follows dependencies [26]. Heuristic Miner allows OR-joins and OR-splits on filtered directly-follows dependencies.

In addition, in the context of Petri nets, researchers have been looking at the region theory to construct a system model from a description of its behavior. State-based regions have been used to construct a Petri net from a transition system as an intermediate representation [21] having filters integrated in the transitions system generation. Language-based regions have been also used to construct a Petri net from a prefix-closed language [11]. In any case, all these algorithms still load the whole event log to build the internal data structure elaborated to discover the final process models, although they start integrating built-in filtering mechanisms operating on the internal data structure.

The preprocessing of event logs has recently gained attention as a useful phase prior to application of process discovery algorithms. Various cleaning techniques are analyzed in [42] as a means to handle noise produced by the presence of infrequent behaviors (outliers) in the event raw data. In fact, these outliers may lead to discover process models exhibiting infrequent execution paths that clutter the model. In particular, outliers may have a negative effect on the precision of the discovered models, as well as on their complexity. To this regard, in [8] the authors show that leaving outliers in event data may have a detrimental effect on the quality of the process models produced by various process discovery algorithms. To detect and remove infrequent process behaviors in raw event data, various filtering methods [8, 38, 39] have been formulated in recent literature. A filtering method for discovering more precise process models in the presence of chaotic activities is also described in [44]. Filtering methods effectively reduce the size of traces elaborated by process discovery algorithms. However, as discussed in [16], the time spent for applying filtering methods is sometimes longer than the time required for discovering a process model from the initial event log. In addition, several filtering methods may have no accurate control over the size of the reduced event log.

Sampling methods reduce the number of traces in event logs. A random trace-based sampling method is described in [3]. This method assumes that traces have different behavior if they have different sets of directly follows relations. A very recent study [37, 16] investigates the effectiveness of applying sampling on event data prior to invoking process discovery algorithms, instead of using all the available event data. The authors compare different biased sampling strategies (frequency-based sampling, length-based sampling, similarity-based sampling and structure-based sampling) and analyze their ability to improve the scalability of the process discovery algorithm. Finally, in [36] the authors propose an iterative trace selection algorithm based on clustering and conformance artifacts. Each iterative phase first performs a trace clustering step with the k-medoids algorithm and the edit distance, in order to populate a sublog with the cluster medoids (i.e., the traces of the original log that are the closest to the cluster centroids). Subsequently, it discovers a process model from this sublog and computes the conformance artifacts of the original log on the discovered process model, in order to identify deviating traces. The iterative phase is repeated on the deviating traces until the quality of the discovered process model improves. As a measure of the quality of a process model, the authors of [36] adopt the F-measure of fitness (computed as Replay fitness [4]) and precision (computed as alignment-based ETC precision [1]). The experiments illustrated in [36] prove that the use of cluster prototypes as input to the process discovery algorithms often improves the F-measure of the discovered process models outperforming both random and biased sampling strategies.

Both filtering and sampling mechanisms reported above down-size event logs by applying some extractive strategy. Differently, the method presented in this study takes advantage of an abstraction strategy to generate new representative traces potentially unobserved in the initial event logs. These new traces may combine different activity sub-sequences also belonging to multiple traces result-

$\langle a, b, c, d, e, f, l, h\rangle^{2000}$
$\langle a, b, i, c, d, e, f, g, h\rangle^{1000}$
$\langle a, e, f, g, h\rangle^{2000}$
$\langle a, f, g, h\rangle^{2000}$
$\langle a, c, d, e, f, g, h\rangle^{1500}$
$\langle a, b, d, e, f, g, h\rangle^{500}$
$\langle a, c, d, e, f, h\rangle^{500}$
$\langle a, g, c, d, e, f, g, h\rangle^{100}$
$\langle a, g, g, h\rangle^{50}$

(a)

$\langle a, b, c, d, e, f, g, h\rangle$
$\langle a, c, d, e, f, g, h\rangle$
$\langle a, e, f, g, h\rangle$
$\langle a, f, g, h\rangle$

(b)

$\langle a, b, c, d, e, f, l, h\rangle$
$\langle a, e, f, g, h\rangle$
$\langle a, f, g, h\rangle$
$\langle a, c, d, e, f, g, h\rangle$
$\langle a, b, i, c, d, e, f, g, h\rangle$

(c)

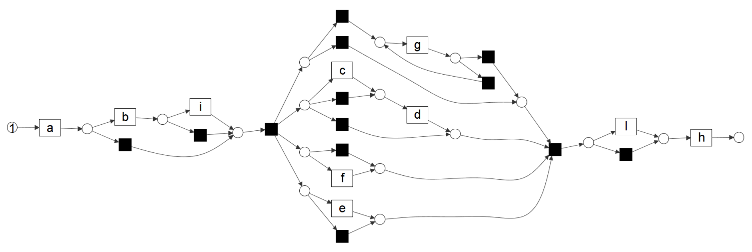
$\langle a, b, c, d, e, f, l, h\rangle$
$\langle a, c, d, e, f, g, h\rangle$
$\langle a, f, g, h\rangle$

(d)

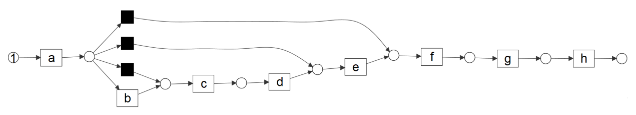
Figure 1: A sample event log (a) and its summaries extracted with PROMISE$^+$ (b), frequency-based SAMPLING [37, 16] (c), CLUSTERING [36] (d).

ing in more flexible event log summaries for the process discovery. In particular, the proposed trace generation approach uses an abstraction process to abstract from the raw events recorded in the initial event logs and construct a predictive process model that can be used to predict the next-activity of a running trace. In this way, we take advantage of a predicting process mining pattern learned from the original event log, in order to improve the process discovery step.

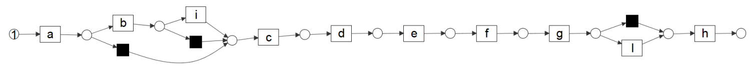
To better illustrate the potential of employing an abstraction-based strategy in the event log summarization, let us consider the event log reported in Figure 1a. The abstraction-based strategy underlying our method generates a new trace $\langle a, b, c, d, e, f, g, h\rangle$ (Figure 1b), which does not belong to the original event log, but stands out as a combination of subsequences coming from multiple original traces (e.g., $\langle \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}, l, h\rangle$, $\langle a, b, i, c, d, e, f, \mathbf{g}, \mathbf{h}\rangle$ and $\langle a, b, d, e, f, \mathbf{g}, \mathbf{h}\rangle$). We note that an effect of the abstraction-based strategy that has led to the generation of this new trace is that the event l has been dropped from the final log summary, while it has been kept in the log summaries extracted with both SAMPLING (Figure 1c) and CLUSTERING (Figure 1d). We compare the process model discovered from the log summary extracted by our method PROMISE$^+$ (Figure 2b) with the original process model (Figure 2a) discovered from the original event log and with the process models discovered from log summaries



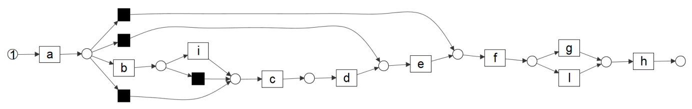
(a)



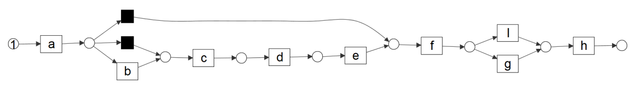
(b)



(c)



(d)



(e)

Figure 2: Petri net representation of (a) original process model; (b) process model discovered with PROMISE$^+$; c) process model discovered with FILTERING; (d) process model discovered with SAMPLING; (e) process model discovered with CLUSTERING. Process models are discovered with Inductive Miner.

extracted by other strategies such as FILTERING (Figure 2c), frequency-based SAMPLING [37, 16] (Figure 2d) and CLUSTERING [36] (Figure 2e). All the process models have been discovered with Inductive Miner. Note that we run FILTERING on 50 different set-up of the embedded filtering parameter of the process discovery algorithm and select the set-up that allows the discovery of the process model achieving the highest F-measure of fitness and precision. We run SAMPLING by ranking all the variant-traces by frequency and selecting the top-variants that allow the discovery of the process model achieving the highest F-measure of fitness and precision. We observe that the abstraction method achieves the highest precision, without significantly decreasing the fitness. It has SAMPLING strategy as the runner-up in terms of F-measure of precision and fitness (as shown by conformance metrics of the compared process models reported in Table 1). In particular, PROMISE$^+$ allows us to discover the process model with the highest precision, while SAMPLING allows us to discover the process model with the highest fitness. In the example log, this behavior mainly depends on the fact that SAMPLING (as well CLUSTERING) keeps the event l in its log summary allowing the discovery of a process model that can parse traces $\langle a, b, c, d, e, f, l, h \rangle$ of the original log. However, in this way, it also parses traces like $\langle a, f, l, h \rangle$ and $\langle a, c, d, e, f, l, h \rangle$ that are not in the original log. Therefore, dropping l from the log summary, PROMISE$^+$ slightly loses in the ability to parsing a few traces in the log, but gains in the ability of not parsing other traces.

Concerning the use of abstraction, we remark that abstraction-based strategies are commonly investigated for text summarizing (see [15] for a survey), but not yet for event log summarizing. To the best of our knowledge, our approach is the first attempt to check the feasibility of an abstraction-based strategy in the event log summarization. On the other hand, various abstraction strategies have already been explored in the process mining field. For example, the authors of [5] describe a means to form abstractions on patterns that capture the manifestation of process model constructs commonly used in event logs. In particular, they present an algorithm for the trace transformation, which replaces the repeated occurrences of the manifestation of a loop with an abstracted activity entity that encodes the notion of a loop. In addition, the transformation algorithm identifies sub processes in traces and replaces them with abstract entities. It also implements abstractions to deal with combinations of choice, parallelism and loops. The authors of [45] introduce a supervised approach that takes advantage of annotations with high-level interpretations of low-level events

Table 1: Conformance metrics of the process models in Fig. 2.

Process model	fitness	precision	F-measure
Original	1.00	0.46	0.63
PROMISE$^+$	0.95	0.97	0.96
FILTERING	0.88	0.93	0.90
SAMPLING	0.99	0.91	0.95
CLUSTERING	0.96	0.92	0.94

to map from fine-granular events to coarse-granular events. A taxonomy of the various strategies to abstract the event log to a higher level of granularity has been recently described in [54]. In addition, the authors of [20] use the notion of log and model abstractions to illustrate a unifying overview of process discovery techniques. In this view, discovery approaches operate on an abstraction of the event log as a means to relate observed behavior to modeled behavior. In any case, to the best of our knowledge, no previous study in the field of process mining has investigated the feasibility of an abstraction strategy that founds on predictive process mining to generate the representative traces of an event log and facilitate the process discovery task.

Finally, we note that the task of trace generation has already received attention in the process mining field [30, 41]. However, these studies describe approaches to generate entire event logs mainly through simulation of some process models. The output of these trace generative approaches is a controlled (big) event log that can be used to evaluate process mining algorithms in a complete and predictable way. Differently, we describe here a method to generate a few new traces from a (big) initial event log, in order to distill a compact version of the most relevant information of the initial log for the process discovery task.

3. Preliminaries

In this paper, we focus on sequences of activities, also called traces, that are combined into event logs and next-activity classification functions that are used in an abstraction strategy to generate event log summaries.

Multisets are commonly used to describe event logs where the same trace may appear multiple times. An event log is a multiset of traces. Each trace describes the life-cycle of a particular case (i.e., a process instance) in terms of the activities executed. In this simple definition of an event log, an event refers to just an activity. Let us consider some set of activities $A \subseteq \mathcal{U}_A$. $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ denotes a sequence over A of length $n = |\sigma|$, where $\sigma(i) = a_i$ for $1 \leq i \leq |\sigma|$. $hd(\sigma, k) = \langle a_1, a_2, \ldots, a_k \rangle$ with $1 \leq k \leq n$ is the head of the sequence consisting of the first k elements. $tl(\sigma, k) = \langle a_{k+1}, a_{k+2}, \ldots, a_n \rangle$ with $1 \leq k \leq n$ is the tail of the sequence composed of the last $|\sigma| - k$ elements. The selection $\sigma(k+1)$ corresponds to the next activity of $hd(\sigma, k)$. Note that the next activity of $hd(\sigma, n)$ is $\perp$ where $\perp$ denotes the end of the trace. For the sequence $\sigma = \langle a, b, c, d, e, f, g, h \rangle$, $\sigma(2) = b$, $hd(\sigma, 2) = \langle a, b \rangle$ and $tl(\sigma, 2) = \langle c, d, e, f, g, h \rangle$. The next activity of $hd(\sigma, 2)$ is c, while the next activity of $hd(\sigma, 8)$ is $\perp$. $\langle\rangle$ is the empty sequence.

Sequences are used to represent traces in an event log. $\sigma_1 \cdot \sigma_2$ is the concatenation of two sequences. In addition, let A^* denote the set of all possible sequences on A. $\mathcal{B}(A^*)$ denotes the set of all multisets over A. For a multiset $L \in B(A^*)$, $L(\sigma)$ is the number of times a distinct variant-sequence σ appears in L. For example, let us consider a set of activities $A = \{a, b, c, e, f, g, h\}$, then $L_1 = []$, $L_2 = [\langle a, b, c, g, h \rangle, \langle a, b, c, d, e, f, g, h \rangle, \langle a, b, c, g, h \rangle]$ and $L_3 = [\langle a, b, c, g, h \rangle^2, \langle a, b, c, d, e, f, g, h \rangle]$ are multisets of $\mathcal{B}(A^*)$. L_1 is the empty multiset, L_2 and L_3 both consist of three sequences and $L_2 = L_3$, i.e., the ordering of sequences is irrelevant and a more compact notation may be used for repeating sequences. Note that both L_2 and L_3 contain three sequences, but two distinct variant-sequences (i.e., $\langle a, b, c, g, h \rangle$ and $\langle a, b, c, d, e, f, g, h \rangle$) as the sequence $\langle a, b, c, g, h \rangle$ appears twice in both L_2 and L_3.

Definition 1 (Trace, Event Log). *Let $A \subseteq \mathcal{U}_A$ be a set of activities. A trace $\sigma \in A^*$ is a sequence of activities. $L \in \mathcal{B}(A^*)$ is an event log, i.e., a multiset of traces.*

We denote as $|L|$ the cardinality (number of traces) of L. We denote as

Table 2: Labeled head sequence multiset T_L extracted from the log L reported in Fig. 1a.

σ	a	occurrences
$\langle a \rangle$	b	3500
$\langle ab \rangle$	c	2000
$\langle abc \rangle$	d	2000
$\langle abcd \rangle$	e	2000
$\langle abcde \rangle$	f	2000
$\langle abcdef \rangle$	l	2000
$\langle abcdefl \rangle$	h	2000
$\langle abcdeflh \rangle$	$\perp$	2000
$\langle ab \rangle$	i	1000
$\langle abi \rangle$	c	1000
$\langle abic \rangle$	d	1000
$\langle abicd \rangle$	e	1000
$\langle abicde \rangle$	f	1000
$\langle abicdef \rangle$	g	1000
$\langle abicdefg \rangle$	h	1000
$\langle abicdefgh \rangle$	$\perp$	1000
$\langle a \rangle$	e	2000
$\langle ae \rangle$	f	2000
$\langle aef \rangle$	g	2000
$\langle aefg \rangle$	h	2000
$\langle aefgh \rangle$	$\perp$	2000
$\langle a \rangle$	f	2000
$\langle af \rangle$	g	2000
$\langle afg \rangle$	h	2000
$\langle afgh \rangle$	$\perp$	2000
$\langle a \rangle$	c	2000
$\langle ac \rangle$	d	2000
$\langle acd \rangle$	e	2000
$\langle acde \rangle$	f	2000
$\langle acdef \rangle$	g	1500
$\langle acdefg \rangle$	h	1500
$\langle acdefgh \rangle$	$\perp$	1500
$\langle ab \rangle$	d	500
$\langle abd \rangle$	e	500
$\langle abde \rangle$	f	500
$\langle abdef \rangle$	g	500
$\langle abdefg \rangle$	h	500
$\langle abdefgh \rangle$	$\perp$	500
$\langle acdef \rangle$	h	500
$\langle acdefh \rangle$	$\perp$	500
$\langle a \rangle$	g	150
$\langle ag \rangle$	c	100
$\langle agc \rangle$	d	100
$\langle agcd \rangle$	e	100
$\langle agcde \rangle$	f	100
$\langle agcdef \rangle$	g	100
$\langle agcdefg \rangle$	h	100
$\langle agcdefgh \rangle$	$\perp$	100
$\langle ag \rangle$	g	50
$\langle agg \rangle$	h	50
$\langle aggh \rangle$	$\perp$	50

max_len_L the maximum length of a trace in L.

Definition 2 (Labeled head sequence multiset). *Let $L \in \mathcal{B}(A^*)$ be an event log. $T_L \in \mathcal{B}(A^* \times A)$ is the multiset of all the head sequences (samples) extracted from the traces of L. Each head sequence is labeled with the next activity (labels) associated to the head sequence in the corresponding trace so that $T_L = [hd(\sigma, k), \sigma(k+1) | \sigma \in L \wedge 1 \leq k \leq |\sigma|]$.*

Let us consider the sample event log reported in Figure 1a. The labeled head sequence multiset extracted form this event log is reported in Table 2.

Definition 3 (Next-activity classification model). *A next-activity classification model F is a function $F\colon A^* \times \mathbb{R}^d \mapsto A$, where d represents the number of real-valued parameters in a model. The last d arguments of the function are later fixed to create a hypothesis function from A^* to A.*

Definition 4 (Next-activity classification hypothesis function). *Let F be a model with d real-valued parameters. Let $\Theta \in \mathbb{R}^d$ be a vector of d real-valued* *parameters. A hypothesis $H_{F,\Theta}$ of the model F is a function: $H_{F,\Theta}\colon A^* \mapsto A$ such that $H_{F,\Theta}(x) \approx F(x, \Theta)$.*

Definition 5 (Cost function). *Let $H_{F,\Theta}$ be a hypothesis of the next-activity classification model function F. The cost function of $H_{F,\Theta}$ is a function $C_{H_{F,\Theta}}\colon A^* \times A \mapsto \mathbb{R}$ such that $C_{H_{F,\Theta}}(x, y)$ measures the penalty of an incorrect classification of the label y done through $H_{F,\Theta}(x)$.*

Definition 6 (Next-activity classification algorithm). *Let T_L be a labeled head sequence multiset. The next-activity classification algorithm determines the hypothesis $H_{F,\Theta}$ that minimizes the cost function $C_{H_{F,\Theta}}$, i.e., such that:*

$$\Theta = \underset{\Theta \in \mathbb{R}^d}{\arg\min} \sum_{(\sigma,a) \in T_L} C_{H_{F,\Theta}}(\sigma, a).$$

The hypothesis $H_{F,\Theta}$ depends on the model type and the labeled multiset. In this study, following the deep learning approach, the model type is the network architecture, i.e., the hypothesis is implicitly determined by the architecture parameters. Under the umbrella of deep learning, the Long-Short-Term Memory (LSTM) neural network [22] is a recurrent network that is suitable to process sequences, such as those underlying a business process event log. The LSTM network uses cyclical connections among its processing units that enable the classification of sequential data by using part of the output of a processing unit for the processing of a new input. The information flows from a unit to another unit with minimal variation, keeping certain aspects constant during the processing of all inputs. This constant input keeps classifications that are coherent over long periods of time. This results in a long-term memory. A common LSTM unit accepts c_{t-1} and h_{t-1} as state information from the prior unrolled cell on the same layer, and x_t as input from cells of the previous layer. In turn, it passes c_t and h_t as new state information to the subsequent unrolled cell on the same layer and also provides h_t as output to the next layer.

Specifically, it performs the following computations:

$$\begin{aligned} f_t &= sigmoid\big(W_f \cdot [h_{t-1}, x_t] + b_f\big), && (1)\\ i_t &= sigmoid\big(W_i \cdot [h_{t-1}, x_t] + b_i\big), && (2)\\ \bar{c}_t &= \tanh\big(W_c \cdot [h_{t-1}, x_t] + b_c\big), && (3)\\ c_t &= (f_t \times c_{t-1}) + (i_t \times \bar{c}_t), && (4)\\ o_t &= sigmoid\big(W_o \cdot [h_{t-1}, x_t] + b_o\big), && (5)\\ h_t &= o_t \times \tanh(c_t), && (6) \end{aligned}$$

where W and b (weights and biases) are the parameters of the network to be learnt. Eq. 1 represents the "forget gate" that determines, based on the inputs x_t and h_{t-1}, which part of the state to forget. Eq. 2 represents the "input gate" and determines which values of the state to update; some of the i_t will be close to zero, others close to one. Eq. 3 computes new candidate values $\bar{c}$. Eq. 2 and 3 determine how the inputs x_t and h_{t-1} contribute to the updated cell state. Eq. 4 computes the new state c_t by adding the information kept in c_{t-1} after forgetting (i.e., $f_t \times c_{t-1}$) to the new information (i.e., $i_t \times \bar{c}_t$). Eq. 5 represents the "output gate" that determines which values of the cell state are provided as output to the following layer and subsequent unrolled cell. Eq. 6 computes the final output h_t as the product between the value provided by the output gate and the tanh of the cell state c_t. The number of LSTM processing units on each layer is a hyper-parameter to fix. The output of the LSTM module is finally fed into a softmax layer, in order to compute the final output (i.e., the next activity) from probabilities of different classes (activities) computed using the *softmax* activation function:

$$y_i = softmax(y)_i = \frac{exp(y_i)}{\sum_j exp(y_j)}. \tag{7}$$

The cross-entropy loss function, that measures the error between the expected label and the probability predicted by the neural network, is commonly used as cost function to be optimized in classification tasks [31].

Several studies [47, 46, 7, 34] have recently proved that LSTM neural network architectures can learn accurate next-activity hypothesis functions $H_{F,\Theta}$ from various event logs. In addition, the authors of [47] have recently shown that repeatedly applying the function $H_{F,\Theta}$ learned by an LSTM neural network, we are able to make accurate longer-term classifications that predict further ahead than a single time step by accurately predicting the tail of a trace given its head.

Definition 7 (Trace tail prediction function). *Let $H_{F,\Theta}\colon A^* \mapsto A$ be a next-activity classification hypothesis function, $\sigma \in A^*$ be a seed sequence, max_len_L be the expected maximum trace length. $H^{\perp}_{F,\Theta}\colon A^* \mapsto A^*$ denotes the function:*

$$H^{\perp}_{F,\Theta}(\sigma) = \begin{cases} \langle\rangle & if\ H_{F,\Theta}(\sigma) = \perp, \\ \langle\rangle & if\ |\sigma| = max\_len_L\ , \\ \langle H_{F,\Theta}(\sigma)\rangle \cdot H^{\perp}_{F,\Theta}\left(\sigma \cdot \langle H_{F,\Theta}(\sigma)\rangle\right) & otherwise \end{cases} \quad (8)$$

that predicts the full continuation of a trace with head σ.

According to Definition 7, we can generate a new trace $\sigma' = \sigma \cdot \langle H^{\perp}_{F,\Theta}(\sigma)\rangle$ by concatenating the head σ and the predicted tail $H^{\perp}_{F,\Theta}(\sigma)$. In principle, any sequence $\sigma \in A^*$ can be used as a seed for this trace generation. However, in this study, we generate new traces from a few seed sequences that appear as heads in real traces of L. Specifically, we use a length-based criterion to determine the seed sequences of the trace generation. Let us consider $HD_l(L) = \{hd(\sigma, l) | \sigma \in L \wedge |\sigma| \geq l\}$, that is, the set of head sequences of L with length equal to l (with $l \geq 1$), we consider all the head sequences of $HD_l(L)$ to prompt the generation of traces to populate a log summary L^*.

Definition 8 (Event log summary). *Let $L \in \mathcal{B}(A^*)$ be an event log, $H^{\perp}_{F,\Theta}\colon A^* \mapsto A^*$ be the trace tail prediction function that repeatedly applies the next-activity classification hypothesis function $H_{F,\Theta}\colon A^* \mapsto A$ learned from T_L, $HD_l(L)$ be the set of head sequences of L with size equal to l. The log summary L^* is a set of traces generated through $H^{\perp}_{F,\Theta}$ from each distinct head sequence $\sigma \in HD_l(L)$, that is, $L^* = \{\sigma \cdot \langle H^{\perp}_{F,\Theta}(\sigma)\rangle | \sigma \in HD_l(L)\}$.*

We highlight that predicting the completion of traces with head in distinct sequences, we can generate distinct traces for L^*. As an example, let us consider the event log L reported in Figure 1a and the next-activity classification hypothesis function $H_{F,\Theta}$ learned through an LSTM neural network from the labeled head sequence multiset T_L reported in Table 2. Let us set $l = 2$. We process the set of 2-sized head sequences $HD_2(L) = \{\langle a, b\rangle, \langle a, c\rangle, \langle a, e\rangle, \langle a, f\rangle\langle a, g\rangle\}$ as seeds for generating the traces of the log summary $L^* = \{\langle a, b, c, d, e, f, g, h\rangle$, $\langle a, c, d, e, f, g, h\rangle$, $\langle a, e, f, g, h\rangle$, $\langle a, f, g, h\rangle$, $\langle a, g, c, d, e, f, g, h\rangle\}$. Note that, as reported in Definition 8, we generate the trace $\langle a, b, c, d, e, f, g, h\rangle$ of L^* as $\langle a, b\rangle \cdot \langle c, d, e, f, g, h\rangle$, where $\langle a, b\rangle \in HD_2(L)$ and $\langle c, d, e, f, g, h\rangle$ is the full continuation that we predict through $H^{\perp}_{F,\Theta}$ for a trace with head $\langle a, b\rangle$ (i.e., $H^{\perp}_{F,\Theta}(\langle a, b\rangle) = \langle c, d, e, f, g, h\rangle$). We similarly generate the traces $\langle a, c, d, e, f, g, h\rangle$, $\langle a, e, f, g, h\rangle$, $\langle a, f, g, h\rangle$ and $\langle a, g, c, d, e, f, g, h\rangle$ of L^* by predicting the completion of traces with head $\langle a, c\rangle$, $\langle a, e\rangle$, $\langle a, f\rangle$ and $\langle a, g\rangle$, respectively.

Further considerations concern the fact that, in the previous example, the log summary L^* comprises the trace $\langle a, g, c, d, e, f, g, h\rangle$ generated by predicting the full continuation of a trace with the head $\langle a, g\rangle$ that is infrequent in L. In fact, the sequence $\langle a, g\rangle$ appears as the head of only 150 traces out of 9650 traces originally stored in L. This trace, if processed by a process discovery algorithm, may lead to discovering process models exhibiting this infrequent execution path that may clutter the model [42]. To handle the possible drawbacks of generating traces that are abstractions of infrequent behaviors in the original log, a sampling post-processing mechanism can be applied to L^* to filter-out generated traces that may reduce the performance of process discovery algorithms.

Definition 9 (Summary event log sampling). *Let L^* be a summary log of L. We define S_{L^*} as trace-based sample of L^* so that $S_{L^*} \subseteq L^*$.*

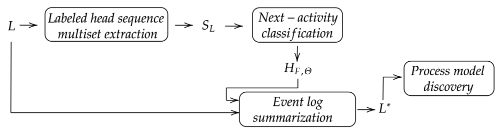

L
Labeled head sequence multiset extraction
S_L
Next – activity classification
$H_{F,\Theta}$
Event log summarization
L^*
Process model discovery

Figure 3: Schema of the PROMISE$^+$ method.

4. Abstraction-based event log summarizing method for process discovery

In this section, we describe PROMISE$^+$. A schematic schematic view of our approach is shown in Figure 3. The method is composed of the following steps:

1. **Labeled head sequence multiset extraction**. It takes as input the initial event log L and returns the labeled head sequence multiset T_L.
2. **Next-activity classification.** It takes as input T_L and learns the next-activity classification hypothesis function $H_{F,\Theta}\colon A^* \mapsto A$.
3. **Event log summarization**. It takes as input both L and $H_{F,\Theta}$ and generates the event log summary L^*.
4. **Process model discovery**. It takes as input L^* and wraps a process discovery algorithm that is applied to the traces of L^*. In this step, the traces of L^* that may decrease the performance of the process discovery algorithm are identified and filtered-out them.

The detailed description of each step is reported in the followings.

4.1. Labeled head sequence multiset extraction

We transform L into T_L that, as presented in Section 3, is composed of all the possible labeled head sequences extracted from all the traces of L. Each head sequence is labeled with the next activity associated with the head sequence in the corresponding trace.

Table 3: Configuration of neural network hyperparameters.

Parameters	Value
Learning Rate	[0.00001, 0.01]
LSTM unit size	{8, 16, 32}
Batch size	[2^5, 2^{10}]

4.2. Next-activity classification

Starting from T_L and using an LSTM neural network as model type, we learn $H_{F,\Theta} \colon A^* \mapsto A$ to classify the next activity of any sequence of activities representing a running trace. As reported in Section 3, this function will be subsequently used as a trace summarizer of L.

We adopt an LSTM neural network architecture with an embedding layer that automatically learns a multi-dimensional real-valued representation of categorical activity sequences. The output of the embedding layer is fed into a recurrent neural network composed of two stacked LSTM layers. The first LSTM layer provides a sequence output to feed the second LSTM layer. We conduct the optimization phase of the hyper-parameters by using 20% of the training set as the validation set. In particular, we perform hyper-parameters optimization with SMAC [23].[1] Table 3 reports the hyper-parameters optimized and the corresponding range of possible values explored with SMAC. The training of the network is accomplished by the Backpropagation algorithm that performs iterative backward passes to find the optimal values of network weights based on gradient descent. We use the cross-entropy loss function for the optimization and perform the Backpropagation training with early stopping to avoid overfitting. We stop the training process when there is no improvement of the loss on the validation set for 20 consecutive epochs. To minimize the loss function, we use the Nadam optimizer. The maximum number of epochs is set to 200.

[1]`https://automl.github.io/SMAC3/master/quickstart.html`

4.3. Event log summarization

Once $H_{F,\Theta}$ has been learned by training an LSTM neural network from T_L, we use it to generate L^* that is an event log summary of L. For this purpose, we first determine the set $HD_l(L)$ of the head sequences of L with size l. These sequences are used as seeds for the trace generation. In fact, for each seed sequence $\sigma \in HD_l(L)$, we repeatedly use $H_{F,\Theta}$ to predict the tail of a new trace having σ as head (according to Def. 7). The new trace is added to L^*. We automatically choose l as the minimum sequence length to select a number of seed sequences greater than 1. As an example, let us consider the event log L in Figure 1a. If we consider $l = 1$, then we get one seed sequence, i.e., the sequence $\langle a \rangle$, to prompt the generation of one trace for L^*. If we consider $l = 2$, then we get five distinct seed sequences, i.e., the sequences $\langle a, b \rangle$, $\langle a, c \rangle$, $\langle a, e \rangle$, $\langle a, f \rangle$ and $\langle a, g \rangle$, to prompt the generation of five distinct traces for L^*. According to the considerations reported above, in the example, we automatically choose $l = 2$.

4.4. Process model discovery

Finally, we discover a process model from L^*. Any process discovery algorithm can be used in this step, although it is recommended to use algorithms that generate sound process models [36]. The selected process discovery algorithm is wrapped within an iterative trace selection process aiming to select the most relevant traces of L^* that contribute to discovering the process model that better conforms to the initial event log L. To this aim, as in [36], we enclose quality assessment metrics by measuring fitness and precision. However, we may integrate any other quality metrics. Both metrics are computed on the traces of the original event log L. As in [36], we consider the classical F-measure to level fitness and precision with equal weights as defined by:

$$F - measure = 2 \times \frac{fitness \times precision}{fitness + precision} \tag{9}$$

We denote by S_{L^*} the sample of L^* that collects the traces contributing to discover the process model that better conforms to the initial event log L. We populate S_{L^*} iteratively by exploring traces of L^* according to a forward trace selection procedure. Starting from an empty S_{L^*} set, we iteratively test the effect of moving a trace from L^* to S_{L^*} by using the F-measure as a quality criterion. At each iterative selection step, we identify the trace $\sigma_{best} \in L^*$ that maximizes the F-measure of the process model that can be discovered from $S_{L^*} \cup \{\sigma_{best}\}$. The trace σ_{best} is definitely moved from L^* to S_{L^*} if and only if the F-measure of the process model discovered from $S_{L^*} \cup \{\sigma_{best}\}$ is greater than the F-measure of process model discovered from S_{L^*}. The forward trace selection procedure stops when all the traces have been moved from L^* to S_{L^*} or the F-measure decreases (i.e., no improvement is obtained).

The pseudo-code of the process discovery step is shown in Algorithm 1. Note that it performs a greedy search that may test $\frac{|L^*|(|L^*|+1)}{2}$ process models, at worst. The result of this search depends on both the quality measure selected to evaluate the process model and the algorithm wrapped for the process discovery. In principle, any quality criterion, as well any process discovery algorithm, may be considered for this step.

Final considerations concern the fact that the described process discovery step may be adopted to explore any set of distinct variant-traces. Therefore, it can be also used to explore the distinct variant-traces of the original event log L, as well as the subset of distinct variant-traces extracted from L with a sampling procedure (e.g., frequency-based sampling).

From this moment on, the baseline configuration that returns the process model discovered from L^* as output is denoted as PROMISE. The upgrade configuration that returns the process model discovered from S_{L^*} as output is denoted as PROMISE$^+$.

5. Experiments

To evaluate the effectiveness of the proposed method, we have conducted a broad range of experiments on several benchmark event logs. The main objective of this experimental study is to investigate the performance of both PROMISE

Algorithm 1: Process discovery step

```
   Data: $L^*$ (event log summary), $L$ (original event log)
   Result: $ProcessModel^*$ (process model)
1  begin
2      $F^* = 0$;
3      $S_{L^*} = \emptyset$
4      $Improvement$ =true
5      while $L^* \neq \emptyset$ and $Improvement$ do
6          $F_{best} = 0$
7          for $\sigma \in L^*$ do
8              $ProcessModel$=process_discovery($S_{L^*} \cup \{\sigma\}$)
9              $F$ = F-measure($ProcessModel, L$);
10             if $F > F_{best}$ then
11                 $F_{best} = F$
12                 $\sigma_{best} = \sigma$
13                 $ProcessModel_{best} = ProcessModel$
14         if $F_{best} > F^*$ then
15             $F^* = F_{best}$
16             $ProcessModel^* = ProcessModel_{best}$
17             $S_{L^*} = S_{L^*} \cup \{\sigma_{best}\}$
18             $L^* = L^* \setminus \{\sigma_{best}\}$
19         else
20             $Improvement$ =false
21     return $ProcessModel^*$
```

and its upgrade PROMISE$^+$ in generating a compact version of event logs prior to invoking process discovery algorithms. Specifically, we aim to answer the following research questions:

445 Q1: Is the proposed method able to improve the quality and complexity of the

discovered process models by also varying the process discovery algorithm wrapped-in?

Q2: How does the proposed method affect the time spent completing the discovery process?

Q3: Does the proposed method outperform related filtering or sampling methods even when also related methods are coupled with the greedy search for improving the quality measure selected for the evaluation?

Q4: Can the use of the proposed method aid the process discovery algorithm in revealing relevant qualitative insights of the original process models?

The implementation of the log summarization method experimented in this study is illustrated in Section 5.1. The benchmark event logs considered in the experiments and the experimental setting are presented in Section 5.2. Finally, the analysis of the results is illustrated in Section 5.3.

5.1. Implementation details

We have implemented both PROMISE and PROMISE$^+$ in Python 3.6.9 – 64 bit version by using Keras 2.3.13 library — a high-level neural network API that adopts TensorFlow 1.15.04 as the back-end. This implementation, that is available via the public GitHub repository,[2] is used to perform the experiments illustrated in this study.

In the experiments illustrated in this study, the next-activity classification hypothesis function is learned through an LSTM neural network. The neural network is trained on a pre-processed version of the original event log L. The pre-processing step is performed to simplify the self-loop events that may appear in each trace $\sigma \in L$. A self-loop is a sub-sequence of a trace where an activity is repeated on n consecutive events with $n \geq 2$. A self-loop with n repetitions is called a n-repetition. Every n-repetition of the same event with a 2-repetition

[2]`https://github.com/vinspdb/PROMISE`

by performing an activity dropping operation. This simplification is introduced to reduce the risk that the LSTM is led to learn a next-activity classification hypothesis function predicting an infinite loop on a specific activity. In addition, as an LSTM neural network architecture trained for sequence classification takes equal-length sequences as input, we use the padding technique in combination with a windowing mechanism [34], in order to standardize different sequence lengths and obtain labeled samples with fixed size. The combination of padding and windowing uses a length equal to W, in order to standardize different sequence lengths and obtain fixed sized sequences. According to this mechanism, dummy events are added to sequences with lengths less than W, while the most recent W activities are kept into sequences with lengths greater than W. We set $W = 4$ in this experimental study.

In addition, we have used three state-of-the-art process discovery algorithms: Inductive Miner [25] and Hybrid ILP Miner [52] (ver 6-10.154) imported from PROM6[3], Split Miner [2] imported from Apromore[4]. We have integrated the implementation of the Replay Fitness [4], the Align-ETConformance [1], the Token-based Repair Generalization [6] and the Petri Net Properties from PM4PY, in order to compute the fitness, precision, generalization and model size (number of transitions, places, and arcs) of the discovered process models, respectively. Finally, we have used the Java plug-in of Show Petri-Net Metrics [24] from PROM6, in order to compute the extended Cardoso index that measures the complexity of a process model by its complex structures, i.e., Xor, Or and And components. The subprocess module[5] present in Python 3.6.9 is adopted, in order to run the Java plug-in of Inductive Miner, Hybrid ILP Miner, Split Miner and Show Petri-Net Metrics through the Python code by creating new processes.

[3]`https://www.promtools.org/doku.php?id=prom610`
[4]`https://apromore.org/platform/tools/`
[5]`https://docs.python.org/3/library/subprocess.html`

Table 4: Event logs description: number of activity classes, traces, events, variants and directly-follows (DF) relations.

Event Log	# Activities	# Traces	# Events	# Variants	# DF Relations
BPIC2012 [12]	23	13087	164506	4336	138
BPIC2018_Insp [14]	15	5485	197717	3190	67
BPIC2019 [13]	42	251734	1595923	11973	538
Hospital [29]	18	100000	451359	1020	143
Road [27]	11	150370	561470	231	70
Sepsis [28]	16	1050	15214	846	115

5.2. Experimental setting

5.2.1. Event logs and process discovery algorithms

We have used six real-life event logs provided by the 4TU Centre for Research. Table 4 reports the characteristics of these event logs. These logs record executions of business processes in healthcare, finance and traffic management. They are heterogeneous in the number of activity classes (from 11 to 42), number of traces (from 1050 to 251734), i number of events (from 15214 to 1595932), number of variants (from 231 to 11973) and number of distinct direct flow relations (from 67 to 537). We have considered these event logs to explore to what extent the method improves the performance of various process discovery algorithms. We have used the following process discovery algorithms: Hybrid ILP Miner [52] (ILP), Inductive Miner [25] (IMi) and Split Miner [2] (SM).

5.2.2. Evaluation metrics

We have analyzed the effectiveness of the proposed method (as well as of the related methods) by evaluating both the complexity and quality of the process models finally discovered, as well as the efficiency of the learning process.

To evaluate the complexity, we have used the model size and the Extended Cardoso index. The model size is measured as the number of transitions, number of arcs and number of places. The Extended Cardoso index [24] is based on the presence of certain splits and joins in the syntactical process definition. In fact, it counts the various splits (XOR, OR, and AND) and give each of them a certain penalty (e.g., the penalty for a place p is the number of subsets of places reachable from a place p).

To evaluate the quality, we have considered the following metrics that are commonly used in process mining literature: fitness, precision, F-measure of fitness and precision, generalization. In this paper, we refer to fitness as the Replay fitness [4], to precision as the alignment-based ETC precision [1] and to generalization as the token-based repair generalization [6]. In particular, the Replay fitness is computed for alignments by returning the percentage of traces that are completely fit, along with a fitness value that is calculated as the average of the fitness values of the single traces. The ETC precision is computed by replaying (whether possible) the different prefixes of the log on the model. At the reached marking, the set of transitions that are enabled in the process model is compared with the set of activities that follow the prefix. The more the sets are different, the lower the precision value is. The more the sets are similar, the higher the precision is.The ETC precision measure is also adopted in the evaluation of the cluster-based sampling procedure described in [37] due to its high performance. However, this precision method works only if the replay of the prefix on the process model works, otherwise the prefix is not considered for the computation of precision. This is a limit of the metric to be taken into account for the analysis of precision results. On the other hand, although various precision measures are formulated in the process mining field, the recent study of [43] have concluded that none of the existing precision measures consistently quantify precision. Finally, the generalization is measured performing a token-based replay operation and computing $1 - avg_t(\sqrt{\frac{1}{freq(t)}})$, where avg_t is the average of the inner value over all the transitions and $freq(t)$ is the frequency of t after the replay. All these metrics have been computed on the original log.

Finally, to measure the efficiency, we have analyzed the computation time spent in minutes to complete each step of the method (i.e., "Labeled head sequence multiset extraction", "Next-activity classification", "Event log summarization" and "Process model discovery"). The computation times have been collected running the experiments on Intel(R) Core(TM) i7-9700 CPU, GeForce RTX 2080 GPU, 32GB Ram Memory, Windows 10 Home.

Note that any other metrics may be used for the evaluation of the quality and complexity of process models. However, the metrics selected in this study have been commonly used in various process mining studies [38, 37, 16] for the evaluation of sampling and filtering approaches.

5.2.3. Compared methods

In these experiments, we have run both the baseline configuration of the proposed method—PROMISE—and its upgrade—PROMISE$^+$. As an initial baseline we have considered ORIGINAL that performs the process discovery algorithm on the original event log without any summarizing mechanism enabled (neither sampling nor filtering). In addition, we have considered the frequency-based sampling method SAMPLING [16] coupled with the F-measure analysis as a related method. For sampling, we have sorted the distinct variant-traces of the original log by frequency and started with the sample composed of the top-frequent variant-trace. We have added top-frequent variants one-by-one to this sample until the F-measure of the process model discovered with the sample increases. As an additional related method, we have evaluated the iterative clustering-based sampling method (CLUSTERING) described in [36]. Note that also CLUSTERING method uses F-measure to extract a sample of trace medoids (selected through an iterative clustering step performed on the original log), which allows the method to improve the F-measure of the process model finally discovered. Note that, as in [36], to elaborate event log summaries, determined through PROMISE, PROMISE$^+$, SAMPLING and CLUSTERING, we have applied the process discovery algorithms by disabling the built-in filtering mechanisms of adopted the process discovery algorithms.

On the other hand, filtering is a prominent approach that various process mining algorithms (comprising Inductive Miner, Hybrid ILP Miner and Split Miner) implement to handle large amounts of events reducing the complexity of the process models, while improving their quality. So, for the completeness of the experiments, we have also applied the process discovery algorithms to the entire event logs by testing the effect of built-in filtering mechanisms in each tested process discovery algorithm (FILTERING), respectively. Experiments with filtering mechanisms have been conducted by selecting the best internal filtering threshold set-up on 50 different configurations. Again, the tested filtering configurations have been evaluated with respect to the F-measure of the process model discovered with each configuration. The configuration achieving the highest F-measure is, finally, selected for the comparative study.

Note that, according to the description reported above, all the methods of this experimental study have been compared coupled with the search of the process model that improves the F-measure. This search may be equally conducted by considering any other process model metric.

5.3. Results and discussion

In this section, we illustrate how the experimental results collected in the evaluation study allow us to address the formulated research questions.

5.3.1. Q1 and Q2

We start analysing the overall performance of PROMISE and PROMISE$^+$ in terms of complexity and quality of process models discovered, as well as time spent completing the discovery process. This analysis is done to quantify the effect of the log summarization performed by coupling the proposed abstraction-based strategy (PROMISE) to the removal of the abstraction-generated traces that may decrease the F-measure of the final process model discovered (PROMISE$^+$). So, as a baseline of this initial evaluation we consider ORIGINAL that discovers process models from initial logs by wrapping the process discovery algorithms with neither the built-in filtering mechanisms nor the sampling pre-processing step. This analysis is conducted to explore how the use of an abstraction-based strategy in log summarizing may be an opportunity to improve the performance of a process discovery algorithm independently of the possible use of

Table 5: Comparison of the process models produced by the ORIGINAL, PROMISE and PROMISE$^+$ approaches in terms of complexity metrics and varying the process discovery algorithm - Hybrid ILP Miner (ILP), Inductive Miner (IMi) and Split Miner (SM).

Configuration		Metric	BPIC2012	BPIC2018_Insp	BPIC2019	Hospital	Road	Sepsis
ILP	Original	generalization	0.98	0.96	0.92	0.92	0.99	0.92
		Model size	33x28x426	26x19x324	61x46x1550	29x22x440	16x15x150	31x20x470
		Cardoso	163	142	561	120	67	209
	PROMISE	generalization	0.99	0.99	0.98	0.98	0.99	0.97
		Model size	18x23x130	9x9x21	19x16x88	13x10x100	11x10x37	12x12x68
		Cardoso	63	27	32	33	17	34
	PROMISE$^+$	generalization	0.99	0.99	0.98	1.00	1.00	0.97
		Model size	17x21x108	9x9x21	8x6x16	9x9x21	8x8x16	12x15x60
		Cardoso	52	27	8	11	8	30
IMi	Original	generalization	0.98	0.93	0.91	0.94	0.99	0.90
		Model size	66x37x140	35x20x72	74x31x154	48x25x98	28x25x70	49x38x114
		Cardoso	52	27	44	37	27	49
	PROMISE	generalization	0.98	0.99	0.98	0.93	0.99	0.96
		Model size	37x26x76	12x12x26	35x25x74	15x6x30	11x7x22	24x25x62
		Cardoso	34	13	31	8	8	28
	PROMISE$^+$	generalization	0.99	0.99	0.97	0.99	1.00	0.97
		Model size	23x17x46	12x12x26	15x10x30	9x7x18	9x7x18	22x18x48
		Cardoso	22	13	12	7	8	22
SM	Original	generalization	0.83	0.80	0.69	0.71	0.82	0.72
		Model size	Model size	84x31x168	568x80x1136	175x37x350	69x24x138	142x35x284
		Cardoso	Cardoso	84	567	174	69	142
	PROMISE	generalization	0.97	0.99	0.97	0.93	0.99	0.89
		Model size	Model size	11x9x22	33x20x66	18x9x36	12x8x24	36x19x72
		Cardoso	Cardoso	10	30	12	9	9
	PROMISE$^+$	generalization	0.99	0.99	0.97	0.99	0.99	0.92
		Model size	Model size	10x9x20	27x19x54	11x9x22	10x8x20	27x18x54
		Cardoso	Cardoso	10	23	10	9	26

well-known extraction-based strategies (e.g., filtering or sampling). In any case, we also compare PROMISE$^+$ to the pipelines with sampling-based pre-processing (SAMPLING and CLUSTERING) and the pipeline with filtering (FILTERING) in the analysis of Q3.

Tables 5 and 6 show the complexity and quality metrics measured on the process models discovered with PROMISE, PROMISE$^+$ and ORIGINAL. The analysis of the complexity metric values shows that both PROMISE and PROMISE$^+$ allow us to discover process models that are simpler than the baseline process models discovered with ORIGINAL. Moreover, we note that the simpler process models discovered with both PROMISE and PROMISE$^+$, as expected, always gain in precision by diminishing the number of unobserved process behaviors (other traces not recorded in the log) that may be inappropriately covered with the process models discovered with ORIGINAL. On the other hand, both PROMISE

Table 6: Comparison of the process models produced by the ORIGINAL, PROMISE and PROMISE$^+$ approaches in terms of quality metrics and varying the process discovery algorithm - Hybrid ILP Miner (ILP), Inductive Miner (IMi) and Split Miner (SM).

Configuration		Metric	BPIC2012	BPIC2018_Insp	BPIC2019	Hospital	Road	Sepsis
ILP	Original	fitness	1.00	1.00	1.00	1.00	1.00	1.00
		precision	0.12	0.13	0.36	0.39	0.53	0.20
		F-measure	0.21	0.22	0.53	0.57	0.69	0.34
	PROMISE	fitness	0.77	0.47	0.72	0.84	0.92	0.77
		precision	0.87	0.95	0.90	0.56	0.93	0.73
		F-measure	0.82	0.63	0.80	0.67	0.92	0.75
	PROMISE$^+$	fitness	0.77	0.47	0.78	0.95	0.91	0.75
		precision	0.88	0.95	1.00	0.99	1.00	0.86
		F-measure	0.82	0.63	0.88	0.97	0.95	0.80
IMi	Original	fitness	1.00	1.00	1.00	1.00	1.00	1.00
		precision	0.14	0.15	0.26	0.56	0.63	0.29
		F-measure	0.25	0.26	0.41	0.72	0.77	0.34
	PROMISE	fitness	0.86	0.71	0.80	0.82	0.95	0.89
		precision	0.75	1.00	0.70	0.54	0.93	0.55
		F-measure	0.80	0.83	0.75	0.65	0.94	0.68
	PROMISE$^+$	fitness	0.82	0.71	0.81	0.90	0.94	0.84
		precision	0.88	1.00	1.00	1.00	1.00	0.81
		F-measure	0.85	0.83	0.90	0.95	0.97	0.82
SM	Original	fitness	0.98	0.97	1.00	1.00	1.00	0.99
		precision	0.46	0.18	0.50	0.75	0.92	0.26
		F-measure	0.63	0.31	0.67	0.86	0.96	0.41
	PROMISE	fitness	0.84	0.66	0.78	0.87	0.95	0.67
		precision	0.90	0.67	1.00	0.93	0.93	0.89
		F-measure	0.87	0.66	0.88	0.90	0.94	0.76
	PROMISE$^+$	fitness	0.84	0.71	0.81	0.94	0.94	0.71
		precision	0.91	1.00	1.00	1.00	1.00	0.95
		F-measure	0.87	0.83	0.90	0.97	0.97	0.81

and PROMISE$^+$ undergo a slight decrease in fitness by discovering process models that may fail in parsing a few traces recorded in the initial event logs. This is commonly accepted, whereas the precision value increases significantly [9]. From this point of view, the F-measure of precision and fitness shows that both PROMISE and PROMISE$^+$ may achieve a better trade-off between precision and fitness in the process models discovered than ORIGINAL. The analysis of the generalization values further supports these conclusions. In fact, both PROMISE and PROMISE$^+$ provide generalization values that are greater than (or equal to) the generalization values measured with ORIGINAL. Both PROMISE and PROMISE$^+$ always enable the discovery of process models that decrease the risk of covering traces that are not in the initial event logs, but are likely to belong to the process that produced the logs.

Further considerations concern the evaluation of how PROMISE$^+$ can actually improve PROMISE in terms quality and complexity of process models discovered. PROMISE$^+$ always outperforms PROMISE by discovering process models that achieve the lower model size and extended Cardoso index, as well as the higher F-measure of fitness and precision, and the higher generalization.

Finally, Figure 4 shows the computation time spent completing the various steps of the compared configurations. These results reveal that both PROMISE and PROMISE$^+$ spend most of their computation time both training the LSTM during the "Next-activity classification" step and discovering the final process model during the "Process model discovery" step. The computation time spent preparing the training data set during the "Labeled head sequence multiset extraction" step, as well as generating the new traces during the "Event log summarization" step is negligible. As expected, PROMISE$^+$ spends more computation time than PROMISE completing the "Process model discovery" step. This is due to the greedy search performed in PROMISE$^+$ for removing traces that, generated during the "Event log summarization" step, may decrease the quality of the process model discovered. Final considerations concern the comparison with ORIGINAL. This configuration spends all its computation time completing the "Process model discovery" step. So, comparing the cumulative time of ORIGINAL, PROMISE and PROMISE$^+$, we note that PROMISE and PROMISE$^+$ are more time-consuming than ORIGINAL. On the other hand, repeating the conclusions drawn above, the additional time spent for the summarization in both PROMISE and PROMISE$^+$ allows us to discover a better process model.

All the conclusions illustrated above are equally drawn independently on the process discovery algorithm and the event log tested.

5.3.2. Q3

We have compared the process models discovered with PROMISE$^+$ to the models discovered with the FILTERING, SAMPLING and CLUSTERING methods. All these methods someway take advantage of the F-measure information in the

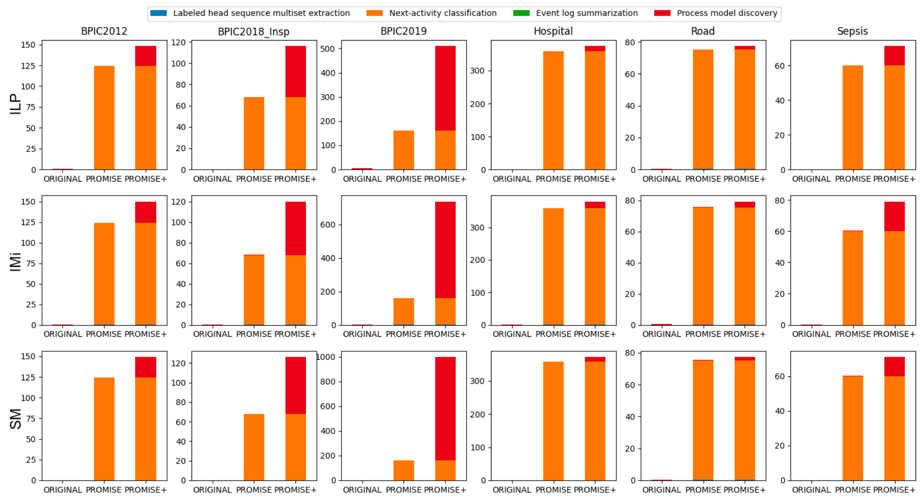


Figure 4: Computation time spent in minutes completing the "Labeled head sequence multiset extraction", "Next-activity classification", "Event log summarization" and "Process model discovery" steps.

discovery of the final process model. Figures 5 and 6 compare the values of the F-measure and extended Cardoso index measured with PROMISE$^+$, FILTERING, SAMPLING and CLUSTERING methods, respectively. To rank the compared methods, we statistically test whether the improvement of both F-measure and extended Cardoso index of the process models discovered with PROMISE$^+$ is significant over the various event logs. To this aim, we have used Friedman's test [10]. This is a non-parametric test that is commonly used to compare multiple methods over multiple event logs. It compares the average ranks of the approaches, so that the best performing approach gets the rank of 1. the second best gets rank 2. The null-hypothesis states that all the methods are equivalent. Under this hypothesis, the ranks of compared methods should be equal. In this study, we reject the null hypothesis with $p\text{-}value \leq 0.05$. As the null-hypothesis has been rejected, that is, no method has been singled out, we have used a post-hoc test—the Nemenyi test—for pairwise comparisons [10].

The results of this test are reported in Figures 7a and 7b for Hybrid ILP Miner (ILP), Figures 7c and 7d for Inductive Miner, (IMi) and Figures 7e and

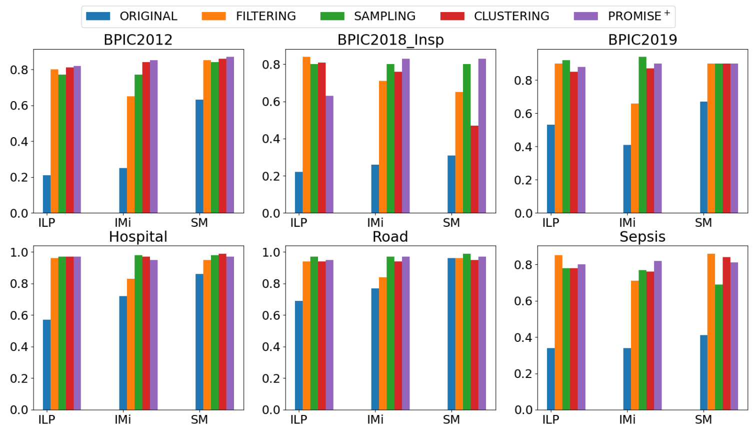


Figure 5: F-measure of precision and fitness: PROMISE$^+$ vs related methods (ORIGINAL, FILTERING, SAMPLING and CLUSTERING) by varying the process discovery algorithm among Hybrid ILP Miner (ILP), Inductive Miner (IMi) and Split Miner (SM).

7f for Split Miner (SM). They show that PROMISE$^+$ enables the discovery of the process models that commonly achieve the highest F-measure by having FILTERING as runner-up with ILP, while SAMPLING as runner-up with IMi and SM. On the other hand, SAMPLING commonly enables the discovery of the simplest process models independently of the process discovery algorithm. So, based upon the previous considerations, SAMPLING is the most relevant competitor of PROMISE$^+$ in this study. In particular, SAMPLING works better than PROMISE$^+$ in event logs with traces distributed according to the Pareto distribution (a large portion of log traces is held by a small fraction of top-frequent variants). For example, SAMPLING works better than PROMISE$^+$ in Hospital, where SAMPLING with IMi selects the top-six frequent variants that cover 87.38% of traces in the event log (see Figure 8a). On the other hand, PROMISE$^+$ works better than SAMPLING in event logs that disregard the Pareto distribution (the majority of traces in the log is spanned on a high number of top-frequent variants). For example, this happens in BPIC2018_Insp (see Figure 8b).

In general, this analysis shows that the proposed abstraction-based strat-

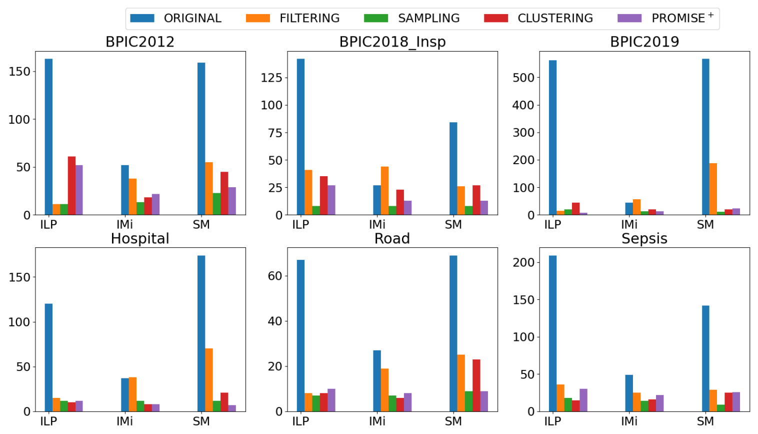
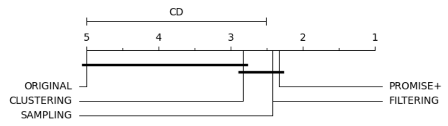
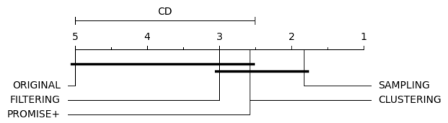


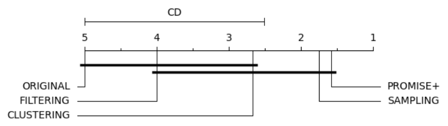
Figure 6: Extended Cardoso index: PROMISE$^+$ vs related methods (ORIGINAL, FILTERING, SAMPLING and CLUSTERING) by varying the process discovery algorithm among Hybrid ILP Miner(ILP), Inductive Miner (IMi) and Split Miner (SM).



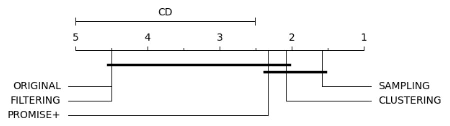
(a) ILP – F-measure



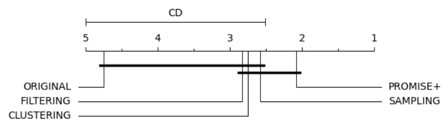
(b) ILP – extended Cardoso



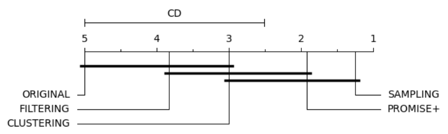
(c) IMi – F-measure



(d) IMi – extended Cardoso

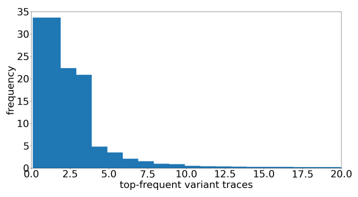


(e) SM – F-measure

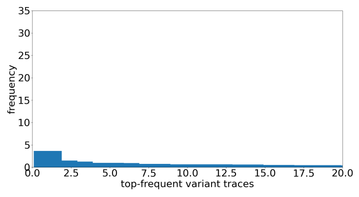


(f) SM – extended Cardoso

Figure 7: Nemenyi test of F-measure and extended Cardoso index on process models discovered using Hybrid ILP Miner (ILP) (a-b), Inductive Miner (IMi) (c-d) and Split Miner (SM) (e-f) with both PROMISE$^+$ and the related methods (ORIGINAL, FILTERING and CLUSTERING). Groups of methods that are not significantly different (at $p \leq 0.05$) are connected.



(a) Hospital



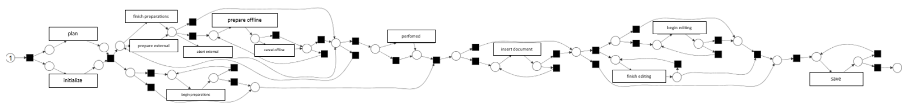
(b) BPIC2018_Insp

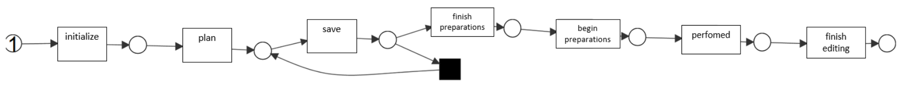
Figure 8: Frequency distribution (axis Y) of 20 - top frequent variant-traces (axis X) in Hospital (Figure 8a) and BPIC2018_Insp (Figure 8b).

egy may be an effective alternative to the extraction-based strategies already formulated in the process discovery field. In this regard, the experiments show that it can summarize the event log by improving the performance of various process discovery algorithms. In any case, there is no summarization method that systematically enables the discovery of the simpler process models with the highest quality in all the event logs. In fact, the performance of the compared methods also depends on the characteristics of the event logs (in addition to the peculiarities of the process discovery algorithms).

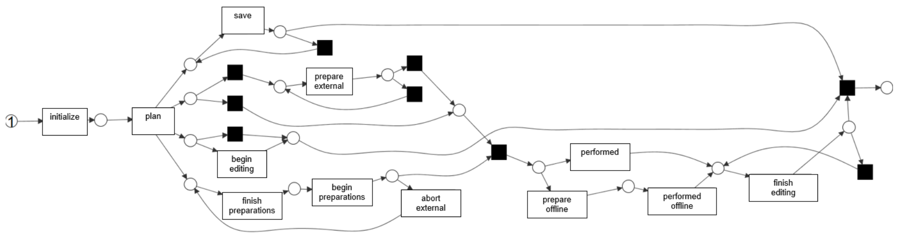
5.3.3. Q4

We complete this study by giving a qualitative understanding of the outcome of the process models discovered with FILTERING, SAMPLING, CLUSTERING and PROMISE$^+$. As an example, let us consider the BPIC2018_Insp event log. Figures 9a, 9b, 9c and 9d show the process models discovered with FILTERING, SAMPLING, CLUSTERING and PROMISE$^+$ from BPIC2018_Insp using Inductive Miner (IMi). In accordance with the conclusions already drawn by analyzing the complexity metrics measured with these methods (Q3), the process model discovered with FILTERING is the most complex, while the process models discovered with both SAMPLING and PROMISE$^+$ are the simplest. On the other hand, the process models discovered with both SAMPLING and PROMISE$^+$ measure the highest F-measure with a precision equal to 1 and a fitness equal to 0.67 and 0.71, respectively. The higher fitness of PROMISE$^+$ is achieved thanks

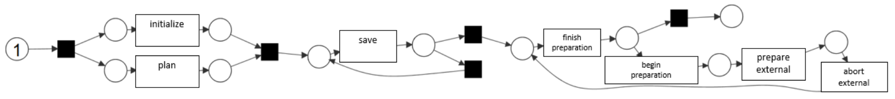

(a) FILTERING - precision=0.63, fitness=0.80, F-measure=0.70, model size=43×31×92, extended Cardoso index = 42.



(b) SAMPLING - precision=1.00, fitness=0.67, F-measure=0.80, model size=8×8×16, extended Cardoso index = 8.



(c) CLUSTERING - precision=0.63, fitness=0.96, F-measure=0.76, model size=21×18×48, extended Cardoso index = 23.



(d) PROMISE$^+$ - precision=1.00, fitness=0.71, F-measure=0.83, model size=12×12×26, extended Cardoso index = 13.

Figure 9: Comparison of process models discovered by IMi with different sampling methods on BPIC2018_Insp.

to the ability of capturing the behavior comprising the sub-sequence of activities "prepare external" and "abort external". This subsequence also appears in the process model discovered with CLUSTERING (that achieves fitness equal to 0.96), but disappear in the process model discovered with SAMPLING.

6. Conclusion

In this paper, we have presented PROMISE$^+$ – a method that generates event log summaries by using an abstraction-based summarization strategy. This strategy leverages a next-activity classification function learned by training a deep neural network architecture composed of LSTM modules. The log summary is then used for the process discovery. Experimental results on different benchmark event logs show that the proposed method provides synthesized logs that enable the discovery of process models with high quality according to the F-measure metric. In particular, the results indicate that PROMISE$^+$ is able to generate proper new traces (that may not appear in the original event log) and the resulting log summaries enable process discovery algorithms to return process models with a good balance between quality measures. In any case, the comparison with the extraction-based counterpart (i.e., SAMPLING) highlights that both approaches are competitive to improve the quality of discovered process models according to the F-measure metric. So, as future work we plan to explore log summarization approaches to other event logs with specific domain knowledge, to better identify which log characteristics contribute more to the success of an extraction or abstraction strategy.

Another important question to address is the selection of the seed sequences for the trace generation in our method. In this study, we use a length-based criterion to select the initial seeds. However, this may lead to select seed sequences that are the head of an infrequent behavior in the event log. We have handled this issue through a process discovery step that performs a greedy search to remove traces generated for the summary that may be outliers. In any case, alternative criteria may be studied to determine seeds by exploring properties like frequency, similarity or structure. The impact of these properties on ranking trace-variants has been recently explored in [16] for sampling.

Also, experimental results have shown that the proposed method provides a good balance between quality and complexity in the process models finally discovered. One main advantage of the method is that the discovered process models are not only precise, but also simple and consequently, easier to understand and explainable. Another advantage is that the process model is discovered from event data generated with a next-activity predictive model. So it may also be seen as a way to provide an easy-to-interpret, graphical explanation of the expected behavior of the predictive model even when it is a black-box model learned by deep neural networks (such as LSTMs). Hence, a direction for future works would be to validate how explainable the resulting process models are for end-users. Indeed the level of explainability of a process model may have a significant impact on its usability. Hence our research can be considered as a step towards adding explanations to business process models, paving the way to create algorithms capable to discover process models with desirable properties of explainability and usability.

Finally, this study opens the door also for different research directions. For example, besides process discovery, the proposed summarization method could be designed for conformance checking.

7. Acknowledgment

The research of Vincenzo Pasquadibisceglie is funded by PON RI 2014-2020 - Big Data Analytics for Process Improvement in Organizational Development - CUP H94F18000270006.

References

[1] Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B. F., & W. van der Aalst (2015). Measuring precision of modeled behavior. *Inf. Syst. E Bus. Manag.*, *13*, 37–67.

[2] Augusto, A., Conforti, R., Dumas, M., La Rosa, M., & Polyvyanyy, A. (2019). Split miner: automated discovery of accurate and simple business process models from event logs. *Knowl. Inf. Syst.*, *59*, 251–284.

[3] Bauer, M., Senderovich, A., Gal, A., Grunske, L., & Weidlich, M. (2018). How much event data is enough? a statistical framework for process discovery. In J. Krogstie, & H. A. Reijers (Eds.), *Advanced Information Systems Engineering* (pp. 239–256). Cham: Springer International Publishing.

[4] Berti, A., & Aalst, W. (2019). Reviving Token-based Replay: Increasing Speed While Improving Diagnostics. In *Proceedings of the International Workshop on Algorithms and Theories for the Analysis of Event Data (ATAED 2019)* (pp. 87–103). volume 2371 of *CEUR Workshop Proceedings*.

[5] Bose, R. P. J. C., & W. van der Aalst (2009). Abstractions in process mining: A taxonomy of patterns. In U. Dayal et al. (Ed.), *Business Process Management, 7th International Conference, BPM 2009, Proceedings* (pp. 159–175). Springer volume 5701 of *LNCS*.

[6] Buijs, J., Dongen, van, B., & Aalst, van der, W. (2014). Quality dimensions in process discovery : the importance of fitness, precision, generalization and simplicity. *International Journal of Cooperative Information Systems*, *23*, 1440001/1–39. doi:`10.1142/S0218843014400012`.

[7] Camargo, M., Dumas, M., & Rojas, O. G. (2019). Learning accurate LSTM models of business processes. In T. T. Hildebrandt et al. (Ed.), *Business Process Management - 17th International Conference, BPM 2019, Proceedings* (pp. 286–302). Springer volume 11675 of *LNCS*.

[8] Conforti, R., Rosa, M. L., & t. Hofstede, A. H. M. (2017). Filtering out infrequent behavior from business process event logs. *IEEE Transactions on Knowledge and Data Engineering*, *29*, 300–314.

[9] De Weerdt, J., De Backer, M., Vanthienen, J., & Baesens, B. (2011). A robust f-measure for evaluating discovered process models. In *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)* (pp. 148–155).

[10] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, *7*, 1–30.

[11] van derWerf, J. M. E. M., van Dongen, B. F., Hurkens, C. A. J., & Serebrenik, A. (2009). Process discovery using integer linear programming. *Fundam. Inf.*, *94*, 387–412.

[12] van Dongen, B. (2012). BPI challenge 2012. doi:`10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`.

[13] van Dongen, B. (2019). BPI challenge 2019. doi:`10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1`.

[14] van Dongen, B., & Borchert, F. F. (2018). BPI challenge 2018. doi:`10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972`.

[15] El-Kassas, W. S., Salama, C. R., Rafea, A. A., & Mohamed, H. K. (2021). Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, *165*, 113679.

[16] Fani Sani, M., van Zelst, S., & W. van der Aalst (2021). The impact of biased sampling of event logs on the performance of process discovery. *Computing*, (pp. 1–20).

[17] W. van der Aalst (2009). Process-aware information systems: Lessons to be learned from process mining. In K. Jensen, & W. van der Aalst (Eds.), *Transactions on Petri Nets and Other Models of Concurrency II: Special Issue on Concurrency in Process-Aware Information Systems* (pp. 1–26). Berlin, Heidelberg: Springer Berlin Heidelberg.

[18] W. van der Aalst (2010). Process discovery: Capturing the invisible. *IEEE Computational Intelligence Magazine*, *5*, 28–41.

[19] W. van der Aalst (2016). *Process Mining - Data Science in Action, Second Edition*. Springer.

[20] W. van der Aalst (2018). Process discovery from event data: Relating models and logs through abstractions. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, *8*.

[21] W. van der Aalst, Rubin, V. A., Verbeek, H. M. W., van Dongen, B. F., Kindler, E., & Günther, C. W. (2010). Process mining: a two-step approach to balance between underfitting and overfitting. *Softw. Syst. Model.*, *9*, 87–111.

[22] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*, 1735–1780.

[23] Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In C. A. C. Coello (Ed.), *Learning and Intelligent Optimization - 5th International Conference, LION 2011, Selected Papers* (pp. 507–523). Springer volume 6683 of *LNCS*.

[24] Lassen, K. B., & W. van der Aalst (2009). Complexity metrics for workflow nets. *Information and Software Technology*, *51*, 610–626.

[25] Leemans, S. J. J., Fahland, D., & van der Aalst, W. (2013). Discovering block-structured process models from event logs - a constructive approach. In J.-M. Colom, & J. Desel (Eds.), *Application and Theory of Petri Nets and Concurrency* (pp. 311–329). Berlin, Heidelberg: Springer Berlin Heidelberg.

[26] Leemans, S. J. J., Fahland, D., & van der Aalst, W. (2013). Discovering block-structured process models from event logs containing infrequent behaviour. In N. Lohmann et al. (Ed.), *Business Process Management Workshops - BPM 2013 International Workshops, Revised Papers* (pp. 66–78). Springer volume 171 of *LNBIP*.

[27] de Leoni, M. M., & Mannhardt, F. (2015). Road traffic fine management process. doi:10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

[28] Mannhardt, F. (2016). Sepsis cases - event log. doi:`10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460`.

[29] Mannhardt, F. (2017). Hospital billing - event log. doi:`10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741`.

[30] Mitsyuk, A. A., Shugurov, I. S., Kalenkova, A. A., & W. van der Aalst (2017). Generating event logs for high-level process models. *Simulation Modelling Practice and Theory*, *74*, 1–16.

[31] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

[32] Pasquadibisceglie, V., Appice, A., Castellano, G., & Malerba, D. (2019). Using convolutional neural networks for predictive process analytics. In *2019 International Conference on Process Mining (ICPM)* (pp. 129–136).

[33] Pasquadibisceglie, V., Appice, A., Castellano, G., & Malerba, D. (2020). Predictive process mining meets computer vision. In D. Fahland et al. (Ed.), *Business Process Management Forum - BPM Forum 2020, Proceedings* (pp. 176–192). Springer volume 392 of *LNBIP*.

[34] Pasquadibisceglie, V., Appice, A., Castellano, G., & Malerba, D. (2021). A multi-view deep learning approach for predictive business process monitoring. *IEEE Transactions on Services Computing (2021)*, .

[35] Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D., & Modugno, G. (2020). ORANGE: outcome-oriented predictive process monitoring based on image encoding and cnns. *IEEE Access*, *8*, 184073–184086.

[36] Sani, M. F., Boltenhagen, M., & W. van der Aalst (2020). Prototype selection using clustering and conformance metrics for process discovery. In A. del-Río-Ortega et al. (Ed.), *Business Process Management Workshops - BPM 2020 International Workshops, Revised Selected Papers* (pp. 281–294). Springer volume 397 of *LNBIP*.

[37] Sani, M. F., van Zelst, S. J., & van der Aalst, W. (2020). Improving the performance of process discovery algorithms by instance selection. *Comput. Sci. Inf. Syst.*, *17*, 927–958.

[38] Sani, M. F., van Zelst, S. J., & W. van der Aalst (2017). Improving process discovery results by filtering outliers using conditional behavioural probabilities. In E. Teniente, & M. Weidlich (Eds.), *Business Process Management Workshops - BPM 2017 International Workshops, Revised Papers* (pp. 216–229). Springer volume 308 of *LNBIP*.

[39] Sani, M. F., van Zelst, S. J., & W. van der Aalst (2018). Applying sequence mining for outlier detection in process mining. In H. Panetto et al. (Ed.), *On the Move to Meaningful Internet Systems. OTM 2018 Conferences, Proceedings, Part II* (pp. 98–116). Springer volume 11230 of *LNCS*.

[40] Sani, M. F., van Zelst, S. J., & W. van der Aalst (2019). The impact of event log subset selection on the performance of process discovery algorithms. In T. Welzer et al. (Ed.), *New Trends in Databases and Information Systems, ADBIS 2019 Short Papers, Proceedings* (pp. 391–404). Springer volume 1064 of *Communications in Computer and Information Science*.

[41] Skydanienko, V., Francescomarino, C. D., Ghidini, C., & Maggi, F. M. (2018). A tool for generating event logs from multi-perspective declare models. In W. van der Aalst et al. (Ed.), *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018* (pp. 111–115). volume 2196 of *CEUR Workshop Proceedings*.

[42] Suriadi, S., Andrews, R., ter Hofstede, A., & Wynn, M. (2017). Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. *Information Systems*, *64*, 132–150.

[43] Tax, N., Lu, X., Sidorova, N., Fahland, D., & W. van der Aalst (2018). The imprecisions of precision measures in process mining. *Information Processing Letters*, *135*, 1–8.

[44] Tax, N., Sidorova, N., & van der Aalst, W. (2019). Discovering more precise process models from event logs by filtering out chaotic activities. *J. Intell. Inf. Syst.*, *52*, 107–139.

[45] Tax, N., Sidorova, N., Haakma, R., & W. van der Aalst (2018). Event abstraction for process mining using supervised learning techniques. In Y. Bi et al. (Ed.), *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016* (pp. 251–269). Springer International Publishing.

[46] Tax, N., Teinemaa, I., & van Zelst, S. J. (2020). An interdisciplinary comparison of sequence modeling methods for next-element prediction. *Software and Systems Modeling*, (pp. 1619–1374).

[47] Tax, N., Verenich, I., La Rosa, M., & Dumas, M. (2017). Predictive business process monitoring with LSTM neural networks. In E. Dubois, & K. Pohl (Eds.), *International Conference on Advanced Information Systems Engineering* (pp. 477–492). Springer.

[48] van der Aalst, W., Weijters, T., & Maruster, L. (2004). Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, *16*, 1128–1142.

[49] vanden Broucke, S. K., & De Weerdt, J. (2017). Fodina: A robust and flexible heuristic process discovery technique. *Decision Support Systems*, *100*, 109–118. Smart Business Process Management.

[50] Weijters, A., & Ribeiro, J. (2011). Flexible heuristics miner (fhm). (pp. 310–317).

[51] van Zelst, S., van Dongen, B., & van der Aalst, W. (2018). Event stream-based process discovery using abstract representations. *Knowl. Inf. Syst.*, *54*, 407–435.

[52] van Zelst, S. J., van Dongen, B. F., & W. van der Aalst (2015). Avoiding over-fitting in ILP-based process discovery. In H. R. Motahari-Nezhad et

al. (Ed.), *Business Process Management* (pp. 163–171). Cham: Springer International Publishing.

[53] van Zelst, S. J., van Dongen, B. F., W. van der Aalst, & Verbeek, H. M. W. (2018). Discovering workflow nets using integer linear programming. *Computing*, *100*, 529–556.

[54] van Zelst, S. J., Mannhardt, F., de Leoni, M., & Koschmider, A. (2020). Event abstraction in process mining: literature review and taxonomy. *Granular Computing*, (pp. 1–18).