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 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)

Preprint submitted to Information Sciences (accepted Version) Published version: https://doi.org/10.1016/j.ins.2022.05.052

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 and enterprise information systems. In principle, these systems are driven by process models 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 [16]. In addition, even when processes are documented in some way, they are often described using a variety of notations [16]. Fortunately, more and more detailed information about the execution of process instances (e.g., activities being executed) are recorded in the form of event logs [17] that enable process discovery algorithms [16].

Process discovery is a branch of process mining [17] that is concerned with the automatic discovery of process models from event logs. Existing process discovery algorithms are formulated in process mining [17] 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 [3].

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 and noisy data settings, where the event data

may be too big to be entirely processed with standard hardware in a reasonable time [46], while the presence of many infrequent behaviors [38] can lead to spaghetti-like models. Recent studies [35, 36] have shown that a straightforward solution to overcome these problems is to build summaries of huge and possibly noisy event logs by down-sizing the amount of event data to be processed with a process discovery algorithm.

To this aim, various filtering [7, 34, 35, 40] and sampling [15, 33, 36] methods have been already proposed in the recent literature as a preprocessing step for process discovery. In addition, several process discovery algorithms [3, 44, 45, 48] have been designed to use filtering in advance to discovering a process model. Notably all these studies 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 face the challenge of properly selecting important traces to be used as input for process discovery by exploring how an abstraction-based approach can be a valid alternative to extractive strategies 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. Our idea is borrowed from the text summarization field (see [14] for a survey), where abstraction-based methods are proved to be commonly more effective than extraction-based methods for realizing text summaries thanks to their ability of combining different word (activity) sub-sequences also belonging to multiple sentences (traces).

So far, various abstraction strategies have already been explored in various fields of process mining, e.g., to replace groups of events with higher-level activities [49] or to provide a unifying overview of process discovery techniques [18]. To the best of our knowledge, no previous study in the field of process mining has investigated the feasibility of an abstraction strategy to generate a summary of an event log and facilitate the process discovery task. In particular, we propose an abstraction-based method that learns a generative representation (abstraction) of an event log that can be helpful to capture prominent process information possibly spread across traces. This abstraction is used for gener-

ating a few "new" representative traces, potentially unobserved in the original log, which convey 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 goes in some way into the direction of [32], where clustering is used to group traces, and cluster medoids are selected as representative, existing traces to populate the log summary. Differently from [32], our method generates a log summary by creating new representative traces, apart from extracting existing significant ones. In particular, new traces, generated through abstraction, may combine different activity sub-sequences also belonging to multiple traces and result in flexible event log summaries for the process discovery. Our expectation is that the abstraction may be particularly beneficial in event logs like BPIC 2018 [13], where the majority of traces in the log is spanned on a high number of top-frequent variants. This condition is opposite to that of event logs like Hospital [27], where a large portion of log traces is held by a small fraction of top-frequent variants according to the Pareto principle. We expect that when the trace distribution does not follow the Pareto principle, the ability of creating new traces by combining activity sub-sequences from multiple, equally distributed traces may contribute to better facilitate the process discovery than extracting only few traces or excerpts of specific traces. In our approach, the new traces generated through the abstraction 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.

In particular, the key idea of our method named PROMISE$^+$ (coupling Predictive pROcess MIning to 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 [6, 29, 30, 31, 41]. Following this research stream, we use a deep neural network with Long Short-Term Memory (LSTM)

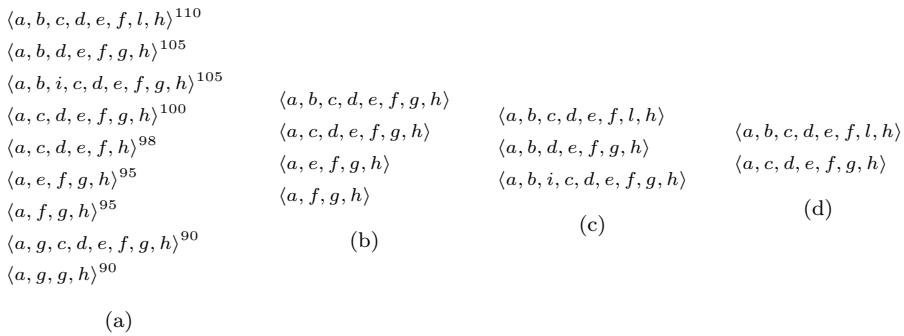


Figure 1: A sample event log, where each superscript number denotes the number of times the distinct variant-trace appears in the event log (a). Summaries extracted from the event log with PROMISE$^+$ (b), frequency-based SAMPLING [15, 33] (c) and CLUSTERING [32] (d).

layers 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, which contribute to maximizing the quality of the process model discovered.

The paper is organized as follows. Section 2 provides a motivating example of this research. Section 3 overviews the related work. Section 4 reports preliminary concepts, while Section 5 describes the proposed PROMISE$^+$ method. In Section 6, we present an extensive empirical study that compares PROMISE$^+$ to sampling and filtering baselines taken from the recent literature on process mining. Finally, Section 7 draws conclusions and sketches the future work.

2. Motivating example

To better illustrate the potential of employing an abstraction-based strategy for event log summarization, let us consider as an example the event log reported in Fig. 1a that contains nine distinct variant traces occurring with

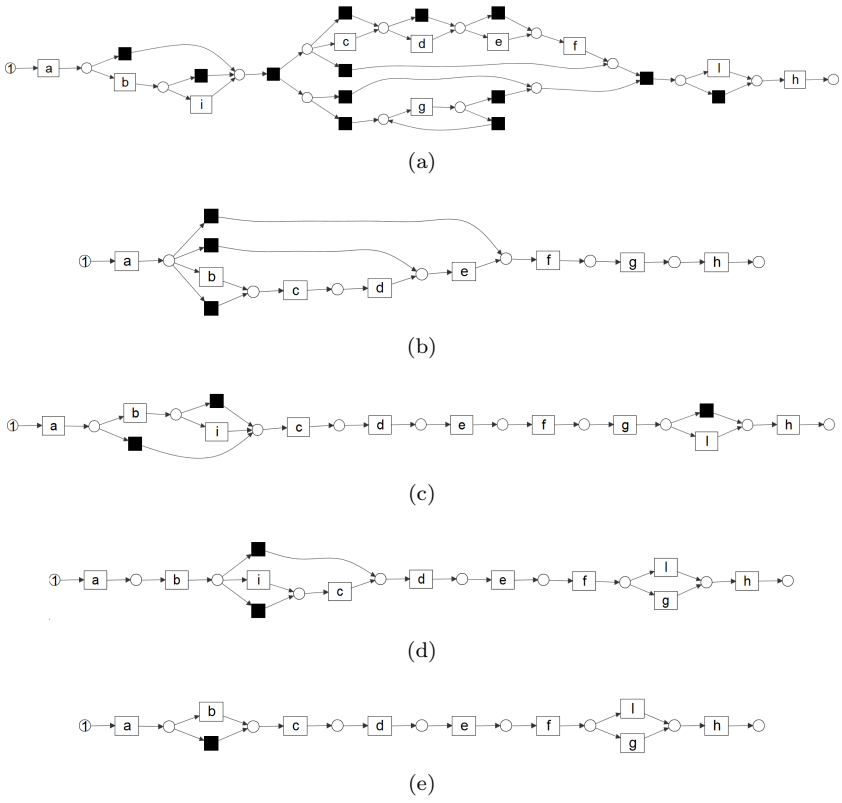


Figure 2: Petri net representation of (a) ORIGINAL process model and process models discovered from log summaries extracted by (b) PROMISE$^+$; (c) FILTERING; (d) SAMPLING; and (e) CLUSTERING.

approximately equally-distributed frequencies. We compared the log summary produced by PROMISE$^+$ (Fig. 1b) with log summaries produced by SAMPLING [15, 33] (Fig. 1c) and CLUSTERING [32] (Fig. 1d). SAMPLING applies the frequency-based sampling strategy, while CLUSTERING groups traces in clusters and selects the cluster medoids. Fig. 1b shows that PROMISE$^+$ generates the traces $\langle a, c, d, e, f, g, h\rangle$ $\langle a, e, f, g, h\rangle$ and $\langle a, f, g, h\rangle$, that correspond to behaviours (variant-traces) belonging to the original event log. Hence, our abstraction strategy sometimes works as a sampling approach. However, PROMISE$^+$ also generates a new trace $\langle a, b, c, d, e, f, g, h\rangle$ that does not belong to the original

event log, but stands out as a combination of subsequences coming from multiple original traces (e.g., $\langle \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}, l, h \rangle$, $\langle a, b, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h} \rangle$, $\langle a, b, i, c, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h} \rangle$) that, otherwise, would disappear from the summary. Conversely, the extractive strategy of SAMPLING and CLUSTERING selects only few variant-traces of the original event log without combining multiple variant-traces in new traces.

A further result of the proposed abstraction strategy is that both i and l, which are respectively the less frequent activities in the event log, are dropped from the final summary, while they are kept in the summary extracted by SAMPLING. Likewise, CLUSTERING keeps event l in the summary. This highlights that PROMISE$^+$ can identify and drop infrequent activity sub-sequences, thus reducing the number of traces to be processed for the final process model discovery. From this point of view, our method may combine the effects of both sampling (by dropping traces) and filtering (by dropping events) thanks to the abstraction mechanism.

Finally, we apply *Inductive Miner* to discover process models from the original event log and from the log summary extracted respectively by PROMISE$^+$, FILTERING, and also CLUSTERING. Note that FILTERING uses the filtering mechanism available on-demand in Inductive Miner to select excerpts of original traces by dropping events. All the process models are represented as Petri nets in Fig. 2. We observe that all the compared summarization methods reduce the number of hidden transitions, i.e., the number of possible alternative behaviours in the discovered process models. In fact, the process model discovered from the ORIGINAL log comprises 13 hidden transitions, while the process models discovered from log summaries produced by PROMISE$^+$, FILTERING, SAMPLING and CLUSTERING comprise 3, 3, 2 and 1 hidden transitions, respectively. The discovery of these simpler process models shows that the considered summarization methods have higher ability of not parsing other traces (precision), but lower ability of fitting traces in the original event log (fitness) compared to the original process model. Nevertheless, PROMISE$^+$ achieves the highest precision, without significantly decreasing the fitness, as shown by conformance metrics of the compared process models reported in Table 1.

Table 1: Conformance metrics of the process models in Fig. 2. A detailed description of these metrics is reported in Section 6.2.2.

Process model	fitness	precision	Fmeasure
ORIGINAL	1.00	0.52	0.684
PROMISE$^+$	0.91	0.98	0.943
FILTERING	0.87	0.94	0.903
SAMPLING	0.87	0.93	0.899
CLUSTERING	0.88	0.93	0.904

The lower precision of models derived by FILTERING, SAMPLING and CLUSTERING compared to that of PROMISE$^+$ partially depends on the fact that all these summarization methods keep the event l in their log summary allowing the discovery of process models that can parse traces like $\langle a, c, d, e, f, g, l, h\rangle$ (FILTERING), $\langle a, b, i, c, d, e, f, l, h\rangle$ and $\langle a, b, d, e, f, l, h\rangle$ (SAMPLING), and $\langle a, c, d, e, f, l, h\rangle$ (CLUSTERING) that do not belong to the original log. Therefore, dropping l from the log summary, PROMISE$^+$ slightly loses the ability to perfectly parsing a few traces in the log e.g., $\langle a, b, c, d, e, f, l, h\rangle$, but gains in the ability of not parsing multiple other traces.

With regard to the reduction of fitness, no process model discovered with a summarization method perfectly fits the original variant-trace $\langle a, g, g, h\rangle$. On the other hand, only the PROMISE$^+$ process model is still able to perfectly fit the original variant-trace $\langle a, e, f, g, h\rangle$. Two event additions – c and d – are minimally required to align this trace to the trace $\langle a, \mathbf{c}, \mathbf{d}, e, f, g, h\rangle$ parsed by the process model discovered with FILTERING and CLUSTERING. Similarly, two event additions – b and d — are minimally required to align the same trace to the trace $\langle a, \mathbf{b}, \mathbf{d}, e, f, g, h\rangle$ parsed by the SAMPLING process model.

3. Related work

Several process discovery algorithms (e.g., Alpha Miner [42], Inductive Miner [23, 24], Heuristic Miner [45], Fodina [44] and Split Miner [3]) have been proposed in the recent process mining literature. Although Alpha Miner [42] and the basic Inductive Miner [23] 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 [48], Heuristic Miner [45], Fodina [44] and Split Miner [3] 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 [24]. 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 [19] 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 [10]. 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 [38] 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 [7] 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 [7, 34, 35] 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 [40]. Filtering methods effectively reduce the size of traces elaborated by process discovery algorithms. However, as discussed in [15], 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 [4]. This method assumes that traces have different behavior if they have different sets of directly follows relations. A very recent study [15, 33] 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. In [32] 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 [32] adopt the Fmeasure of fitness and precision. The experiments illustrated in [32] prove that the use of cluster prototypes as input to the process discovery algorithms often improves the Fmeasure of the discovered process models outperforming both random and biased sampling strategies.

Finally, we note that the task of trace generation addressed in this paper to perform the log summarization has already received attention in the process mining field [28, 37]. However, these studies describe approaches to generate entire event logs mainly through simulation of some process models. The output of these 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 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. Based upon these considerations, we select filtering and sampling approaches as related methods of the experimental study.

4. 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 $\sigma = \langle a, b, c, d, e, f, l, h \rangle$, $\sigma(2) = b$, $hd(\sigma, 2) = \langle a, b \rangle$ and $tl(\sigma, 2) = \langle c, d, e, f, l, h \rangle$. 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 $A = \{a, b, c, d, e, f, g, h, l\}$, then $L_1 = []$, $L_2 = [\langle a, b, c, g, h \rangle, \langle a, b, c, d, e, f, l, h \rangle, \langle a, b, c, g, h \rangle]$ and $L_3 = [\langle a, b, c, g, h \rangle^2, \langle a, b, c, d, e, f, l, h \rangle]$ are multisets of $\mathcal{B}(A^*)$. L_1 is the empty mul-

tiset, 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 $\langle a, b, c, d, e, f, l, h\rangle$) 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 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 Fig. 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 $C_{H_{F,\Theta}}\colon A^* \times$*

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

σ	next activity	occurrences	σ	next activity	occurrences
$\langle a \rangle$	b	320	$\langle a, c, d, e, f \rangle$	g	100
$\langle a, b \rangle$	c	110	$\langle a, c, d, e, f, g \rangle$	h	100
$\langle a, b, c \rangle$	d	110	$\langle a, c, d, e, f, g, h \rangle$	$\perp$	100
$\langle a, b, c, d \rangle$	e	110	$\langle a, c, d, e, f \rangle$	h	98
$\langle a, b, c, d, e \rangle$	f	110	$\langle a, c, d, e, f, h \rangle$	$\perp$	98
$\langle a, b, c, d, e, f \rangle$	l	110	$\langle a \rangle$	e	95
$\langle a, b, c, d, e, f, l \rangle$	h	110	$\langle a, e \rangle$	f	95
$\langle a, b, c, d, e, f, l, h \rangle$	$\perp$	110	$\langle a, e, f \rangle$	g	95
$\langle a, b \rangle$	d	105	$\langle a, e, f, g \rangle$	h	95
$\langle a, b, d \rangle$	e	105	$\langle a, e, f, g, h \rangle$	$\perp$	95
$\langle a, b, d, e \rangle$	f	105	$\langle a \rangle$	f	95
$\langle a, b, d, e, f \rangle$	g	105	$\langle a, f \rangle$	g	95
$\langle a, b, d, e, f, g \rangle$	h	105	$\langle a, f, g \rangle$	h	95
$\langle a, b, d, e, f, g, h \rangle$	$\perp$	105	$\langle a, f, g, h \rangle$	$\perp$	95
$\langle a, b \rangle$	i	105	$\langle a \rangle$	g	180
$\langle a, b, i \rangle$	c	105	$\langle a, g \rangle$	c	90
$\langle a, b, i, c \rangle$	d	105	$\langle a, g, c \rangle$	d	90
$\langle a, b, i, c, d \rangle$	e	105	$\langle a, g, c, d \rangle$	e	90
$\langle a, b, i, c, d, e \rangle$	f	105	$\langle a, g, c, d, e \rangle$	f	90
$\langle a, b, i, c, d, e, f \rangle$	g	105	$\langle a, g, c, d, e, f \rangle$	g	90
$\langle a, b, i, c, d, e, f, g \rangle$	h	105	$\langle a, g, c, d, e, f, g \rangle$	h	90
$\langle a, b, i, c, d, e, f, g, h \rangle$	$\perp$	105	$\langle a, g, c, d, e, f, g, h \rangle$	$\perp$	90
$\langle a \rangle$	c	198	$\langle a, g \rangle$	g	90
$\langle a, c \rangle$	d	198	$\langle a, g, g \rangle$	h	90
$\langle a, c, d \rangle$	e	198	$\langle a, g, g, h \rangle$	$\perp$	90
$\langle a, c, d, e \rangle$	f	198			

$A \mapsto \mathbb{R}$ *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 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:

$$f_t = sigmoid\big(W_f \cdot [h_{t-1}, x_t] + b_f\big), \quad (1)$$

$$i_t = sigmoid\big(W_i \cdot [h_{t-1}, x_t] + b_i\big), \quad (2)$$

$$\bar{c}_t = \tanh\big(W_c \cdot [h_{t-1}, x_t] + b_c\big), \quad (3)$$

$$c_t = (f_t \times c_{t-1}) + (i_t \times \bar{c}_t), \quad (4)$$

$$o_t = sigmoid\big(W_o \cdot [h_{t-1}, x_t] + b_o\big), \quad (5)$$

$$h_t = o_t \times \tanh(c_t), \quad (6)$$

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 Eq. 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 units on each layer is a hyper-parameter to fix. The output of the LSTM module is finally fed into a softmax layer 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.

Several studies [6, 31, 41] 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 [41] 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:*

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

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 Fig. 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 $\langle a, b, c, d, ef, g, h\rangle$ 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 180 traces out of 888 traces

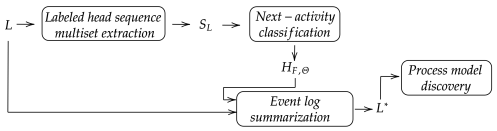


Figure 3: Schema of the PROMISE$^+$ method.

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 [38]. 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 $S_{L^*} \subseteq L^*$.*

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

In this section, we describe PROMISE$^+$. A schematic view of our approach is shown in Fig. 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.

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

5.1. Labeled head sequence multiset extraction

We transform L into T_L. As presented in Section 4, T_L is the multiset of all the possible labeled head sequences extracted from all the traces of L. In particular, for each trace $\sigma \in L$, we extract the $|\sigma|$ head sequences of σ with length ranging between 1 and $|\sigma|$.

Each head sequence is labeled with the next activity associated with the head sequence in the corresponding trace. Note that, as shown in the example in Table 2, the head sequence with length $|\sigma|$, that corresponds to the entire trace, is labeled with the end of the trace $\perp$.

5.2. 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 4, 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 [21].[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

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

Table 3: Configuration of neural network hyperparameters

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

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.

5.3. 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 Fig. 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 $l = 2$.

5.4. 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 [32]. 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 [32], we enclose conformance quality assessment metrics by measuring fitness and precision. However, we may integrate any other quality metrics. Notice that several

fitness and precision metrics may be used. A description of the fitness and precision metrics considered in this study is reported in Section 6.2.2. Both metrics are computed on the traces of the original event log L. As in [32], we consider the classical Fmeasure to level fitness and precision with equal weights as defined by:

$$Fmeasure = 2 \times \frac{fitness \times precision}{fitness + precision} \tag{9}$$

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 Fmeasure as a quality criterion. At each iterative selection step, we identify the trace $\sigma_{best} \in L^*$ that maximizes the Fmeasure 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 Fmeasure of the process model discovered from $S_{L^*} \cup \{\sigma_{best}\}$ is greater than the Fmeasure 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 Fmeasure 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

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*} = ∅
4      Improvement =true
5      while L* ≠ ∅ and Improvement do
6          F_best = 0
7          for σ ∈ L* do
8              ProcessModel=process_discovery(S_{L*} ∪ {σ})
9              F = Fmeasure(ProcessModel, L);
10             if F > F_best then
11                 F_best = F
12                 σ_best = σ
13                 ProcessModel_best = ProcessModel
14         if F_best > F* then
15             F* = F_best
16             ProcessModel* = ProcessModel_best
17             S_{L*} = S_{L*} ∪ {σ_best}
18             L* = L* \ {σ_best}
19         else
20             Improvement =false
21     return ProcessModel*
```

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$^+$.

6. 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 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: (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 forward trace selection for improving the quality measure selected for the evaluation? The implementation of the log summarization method experimented in this study is illustrated in Section 6.1. The benchmark event logs considered in the experiments and the experimental setting are presented in Section 6.2. Finally, the analysis of the results is illustrated in Section 6.3.

6.1. 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

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

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 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 [31], 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 [23] and Hybrid ILP Miner [47] (ver 6-10.154) imported from PROM6[3], Split Miner [3] imported from Apromore[4]. We have integrated the implementation of the Cost-based fitness [1] and the Align-ETConformance [2] from PM4PY. Both metrics have been measured with alignments computed with the algorithm A^* [43]. We have integrated the Token-based Repair Generalization [5] and the Petri Net Properties from PM4PY to compute the generalization and model size (number of transitions, places, and arcs) of the discovered process models, respectively. We have used the Java plug-in of Show Petri-Net Metrics [22] from PROM6 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. We have adopted the subprocess module[5] of Python 3.6.9 to run the Java plug-in of Inductive Miner, Hybrid ILP Miner, Split Miner and Show

[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 [11]	23	13087	164506	4336	138
BPIC2018_Insp [13]	15	5485	197717	3190	67
BPIC2019 [12]	42	251734	1595923	11973	538
Hospital [27]	18	100000	451359	1020	143
Road [25]	11	150370	561470	231	70
Sepsis [26]	16	1050	15214	846	115

Petri-Net Metrics through the Python code by creating new processes.

6.2. Experimental setting

6.2.1. 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 activities classes (from 11 to 42), number of traces (from 1050 to 251734), 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 [47] (ILP), Inductive Miner [23] (IMi) and Split Miner [3] (SM).

6.2.2. 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 [22] 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 computed the following metrics on the original log: fitness, precision, Fmeasure of fitness and precision, generalization. In the following we refer to fitness as the cost-based fitness [1], to precision as the alignment-based ETC precision [2] and to generalization as the token-based repair generalization [5].

- **Fitness.** The cost-based fitness is computed for trace alignments by returning a fitness value that is the average of the fitness values of the single traces. The fitness of a single trace is computed as $\left(1 - \frac{alignment\_cost}{best\_of\_worst\_cost}\right)$, where $alignment\_cost$ is the total cost of having inserted/skipped activities and $best\_of\_worst\_cost$ is the best total cost of considering all events as inserted activities.

- **Precision.** The ETC precision is computed by constructing a prefix automaton that consists of one state per unique head sequence (prefix) of the event log. As reported in [2], we use alignments to calculate the prefix automaton on the aligned event log. For each state in the prefix automaton, we determine which activities are allowed as next activities by the process model. Activities that are allowed as next activities for some head sequence, but that are never observed in the event log after this head sequence are referred to as *escaping edges*. The precision of the event log is computed as $\left(1 - \frac{nonescaping\_edges}{total\_edges}\right)$, where $nonescaping\_edges$ is the weighted number of non-escaping edges and $total\_edges$ is the weighted total number of edges. Weights are defined by the number of occurrences of each state in the multiset of head sequences in the original log. The lower the number of non-escaping edged, the higher the precision. The ETC precision measure is also adopted in the evaluation of the cluster-based sampling procedure described in [33] 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 [39] have concluded that none of the existing precision measures consistently quantify precision.

- **Generalization.** The generalization is measured performing a token-based replay operation and computing $g1 - avg_t\left(\sqrt{\frac{1}{freq(t)}}\right)$, 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.

Finally, to measure the efficiency, we have analyzed the computation time spent in minutes to complete each step of the method. 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.

Of course any other metrics may be used for the evaluation of the complexity and quality of process models. However, the metrics selected in this study have been commonly used in various process mining studies [15, 33, 34] for the evaluation of sampling and filtering approaches.

6.2.3. 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 [15] coupled with the Fmeasure 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 Fmeasure 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 [32]. Note that also CLUSTERING method uses Fmeasure 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 Fmeasure of the process model finally discovered. As in [32], 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 Fmeasure of the process model discovered with each configuration. The configuration achieving the highest Fmeasure is, finally, selected for the comparative study.

So, 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 Fmeasure. This search may be equally conducted by considering any other process model metric.

6.3. 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.

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	Model size	33x28x426	26x19x324	61x46x1550	29x22x440	16x15x150	31x20x470
		Cardoso	163	142	561	120	67	209
	PROMISE	Model size	18x23x130	9x9x21	19x16x88	13x10x100	11x10x37	12x12x68
		Cardoso	63	27	32	33	17	34
	PROMISE$^+$	Model size	17x21x108	9x9x21	8x6x16	9x9x21	8x8x16	12x15x60
		Cardoso	52	27	8	11	8	30
IMi	Original	Model size	66x37x140	35x20x72	74x31x154	48x25x98	28x25x70	49x38x114
		Cardoso	52	27	44	37	27	49
	PROMISE	Model size	37x26x76	12x12x26	35x25x74	15x6x30	11x7x22	24x25x62
		Cardoso	34	13	31	8	8	28
	PROMISE$^+$	Model size	23x17x46	12x12x26	15x10x30	9x7x18	9x7x18	22x18x48
		Cardoso	22	13	12	7	8	22
SM	Original	Model size	159x45x318	84x31x168	568x80x1136	175x37x350	69x24x138	142x35x284
		Cardoso	159	84	567	174	69	142
	PROMISE	Model size	36x23x72	11x9x22	33x20x66	18x9x36	12x8x24	36x19x72
		Cardoso	34	10	30	12	9	9
	PROMISE$^+$	Model size	31x21x62	10x9x20	27x19x54	11x9x22	10x8x20	27x18x54
		Cardoso	29	10	23	10	9	26

6.3.1. Q1 and Q2

We start analysing the 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 decrease the Fmeasure of the final process model discovered (PROMISE$^+$). As a baseline of this 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 the log summarization may be an opportunity to improve the performance of a process discovery algorithm independently of the use of well-known extraction-based strategies (e.g., filtering or sampling). 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 pro-

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
		Fmeasure	0.21	0.22	0.53	0.57	0.69	0.34
		Generalization	0.98	0.96	0.92	0.92	0.99	0.92
	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
		Fmeasure	0.82	0.63	0.80	0.67	0.92	0.75
		Generalization	0.99	0.99	0.98	0.98	0.99	0.97
	PROMISE$^+$	Fitness	0.77	0.47	0.78	0.87	0.88	0.75
		Precision	0.88	0.95	1.00	1.00	1.00	0.86
		Fmeasure	0.82	0.63	0.88	0.93	0.94	0.80
		Generalization	0.99	0.99	0.98	1.00	1.00	0.97
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
		Fmeasure	0.25	0.26	0.41	0.72	0.77	0.34
		Generalization	0.98	0.93	0.91	0.94	0.99	0.90
	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
		Fmeasure	0.80	0.83	0.75	0.65	0.94	0.68
		Generalization	0.98	0.99	0.98	0.93	0.99	0.96
	PROMISE$^+$	Fitness	0.81	0.71	0.81	0.87	0.94	0.84
		Precision	0.90	1.00	1.00	1.00	1.00	0.81
		Fmeasure	0.85	0.83	0.90	0.93	0.97	0.82
		Generalization	0.99	0.99	0.97	0.99	1.00	0.97
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
		Fmeasure	0.63	0.31	0.67	0.86	0.96	0.41
		Generalization	0.83	0.80	0.69	0.71	0.82	0.72
	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
		Fmeasure	0.87	0.66	0.88	0.90	0.94	0.76
		Generalization	0.97	0.99	0.97	0.93	0.99	0.89
	PROMISE$^+$	Fitness	0.84	0.71	0.81	0.87	0.94	0.71
		Precision	0.91	1.00	1.00	1.00	1.00	0.95
		Fmeasure	0.87	0.83	0.90	0.93	0.97	0.81
		Generalization	0.99	0.99	0.97	0.99	0.99	0.92

cess 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, 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 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 [8]. From this point of view, the Fmeasure 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 Fmeasure of fitness and precision, and the higher generalization.

Finally, Fig. 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. On the other hand, ORIGINAL 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. However, the additional time spent for the summarization in both PROMISE and

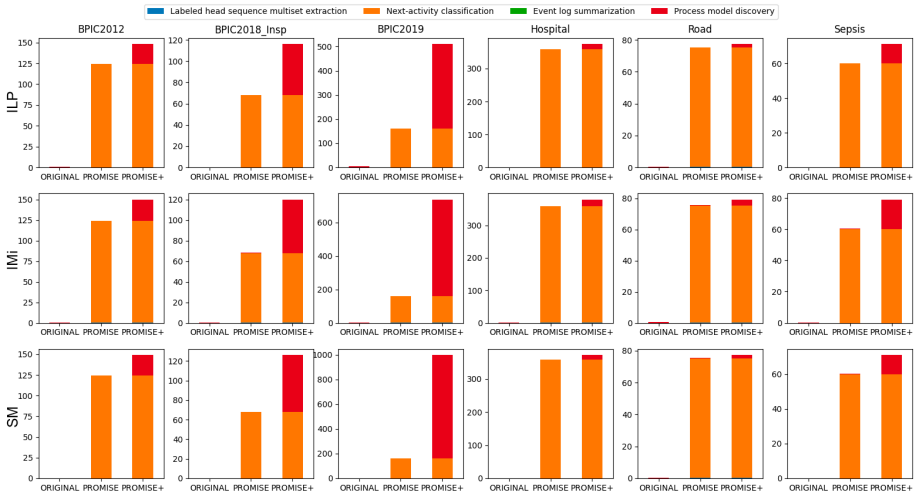


Figure 4: Computation time spent in minutes completing the "Labeled head sequence multiset extraction", "Next-activity classification", "Event log summarization" and "Process model discovery" steps.

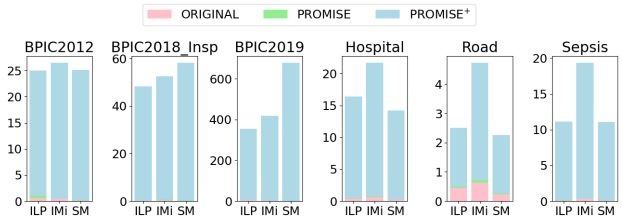


Figure 5: Detail on the computation time spent in minutes by ORIGINAL, PROMISE and PROMISE$^+$ completing the "Process model discovery" step.

PROMISE$^+$ allows us to discover a better process model.

By exploring in depth the computation times, we note that the training of the LSTM is a time consuming task. In any case, the LSTM, that is trained in this study to facilitate the process discovery, is also available for dealing with predictive process monitoring tasks, e.g., to proactively monitor the logged business process and ensure that the business activities will run in a desired manner [31]. On the other hand, the Petri net representation of the process model discovered from the event log summary generated through the LSTM can be seen,

in some way, as a graphical explanation of how the LSTM performs its next-activity predictions. This consideration is inspired by the work in [20], where a Directly-Follows Graph (DFG) was generated from the probability matrix populated with all the probabilities produced by the LSTM trained to predict next activities of the original event log. In fact, authors of [20] used the DFG to graphically explain several decision-making processes of the LSTM model. Finally, as concerns the time spent for the "Process model discovery" step (Fig. 5), we note that PROMISE requires less computation time than ORIGINAL since it takes advantage of a smaller amounts of event data to process. On the other hand, PROMISE$^+$ requires more computation time than PROMISE and ORIGINAL to complete the "Process model discovery" step. This is due to the forward trace selection performed in PROMISE$^+$ for removing traces representing infrequent behaviors that have been possibly generated during the "Event log summarization" step. This removal is essential because leaving these traces may decrease the quality of the process model discovered. Most of the computation time spent completing the forward trace selection is devoted to perform the trace alignments to measure the fitness and precision of the event log with respect to the final process models discovered from the candidate traces explored in the search. However, the improvement achieved in the quality and complexity of the final process models compensates for the increase of time complexity introduced with the forward trace selection. In any case, exploring alternatives to the forward trace selection is a future research direction of this study.

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

6.3.2. 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 Fmeasure information in the discovery of the final process model. Fig. 6 and 7 compare the values of the Fmeasure and extended Cardoso index measured with PROMISE$^+$, FILTERING,

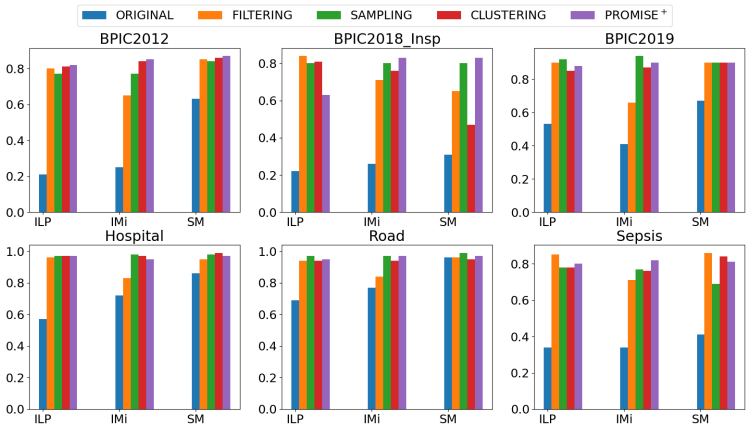


Figure 6: Fmeasure of precision and fitness: PROMISE$^+$ vs related methods (ORIGINAL, FILTERING, SAMPLING and CLUSTERING) by varying the process discovery algorithm among Hybrid ILP Miner (ILP), Inductive Miner (IMi) and Split Miner (SM).

SAMPLING and CLUSTERING methods, respectively. To rank the compared methods, we statistically test whether the improvement of both Fmeasure 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 [9]. 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 [9].

The results of this test are reported in Fig. 8a and 8b for Hybrid ILP Miner (ILP), Fig. 8c and 8d for Inductive Miner, (IMi) and Fig. 8e and 8f for Split Miner (SM). They show that PROMISE$^+$ enables the discovery of the process models that commonly achieve the highest Fmeasure by having FILTERING as runner-up with ILP, while SAMPLING as runner-up with IMi and

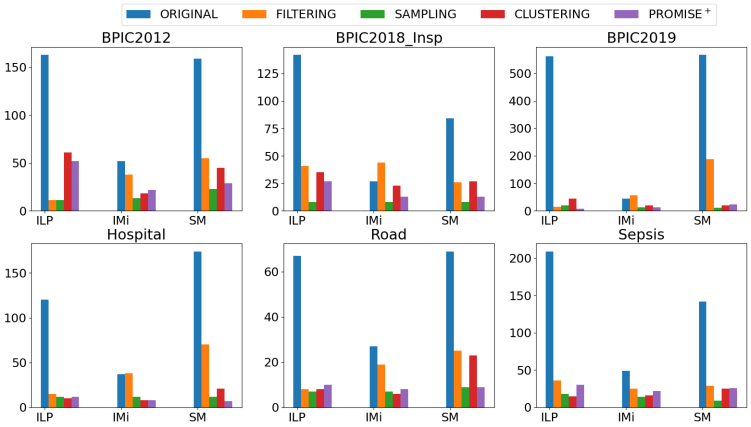
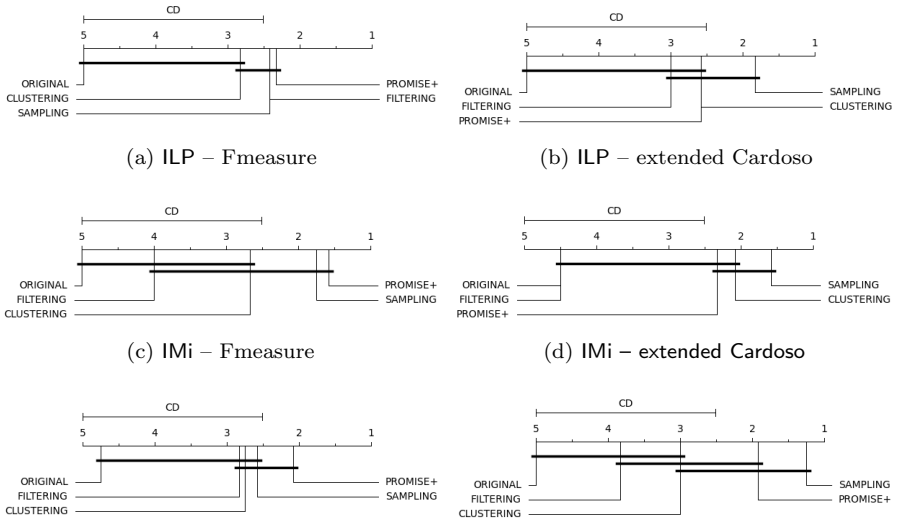


Figure 7: Extended Cardoso index: PROMISE$^+$ vs related methods (ORIGINAL, FILTERING, SAMPLING and CLUSTERING) by varying the process discovery algorithm among Hybrid ILP Miner(ILP), Inductive Miner (IMi) and Split Miner (SM).



(a) ILP – Fmeasure

(b) ILP – extended Cardoso

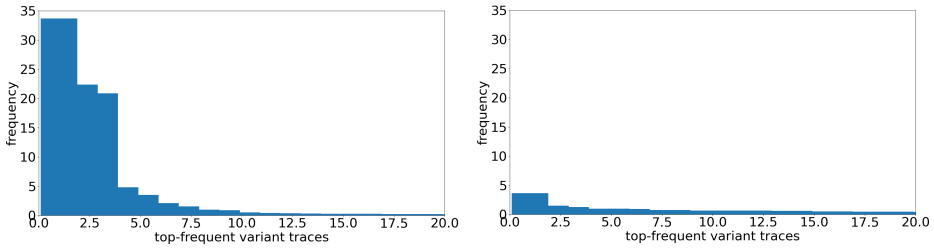
(c) IMi – Fmeasure

(d) IMi – extended Cardoso

(e) SM – Fmeasure

(f) SM – extended Cardoso

Figure 8: Nemenyi test of Fmeasure 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 9: Frequency distribution (axis Y) of 20 - top frequent variant-traces (axis X) in Hospital (Fig. 9a) and BPIC2018_Insp (Fig. 9b).

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 Fig. 9a). 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) as already anticipated in the motivating example of Section 2. For example, this happens in BPIC2018_Insp (see Fig. 9b) when both IMi and SM are selected as process discovery algorithms. On the other hand, in this event log, the Fmeasure of the process model discovered with ILP using the log summary produced by PROMISE$^+$ is lower than the Fmeasure of the process model discovered with ILP using the log summary produced by SAMPLING. However, this performance mainly depends on ILP. In fact, PROMISE$^+$ integrates the process discovery algorithm in the forward trace selection step. In BPIC2018_Insp, this step selects the same two traces for the process discovery

independently of the process discovery algorithm adopted. However, the process model discovered by processing these two traces with ILP is different from the process model discovered by processing the same two traces with IMi or SM. Finally, the process model discovered with IMi and SM on the event log summary generated by PROMISE$^+$ for BPIC2018_Insp achieves the highest Fmeasure of this experiment. This highlights that the performance of the compared methods also depend on the peculiarities of the process discovery algorithms.

In general, this analysis shows that the proposed abstraction-based strategy may be an effective alternative to the extraction-based strategies already formulated in the process discovery field, which makes available a predictive model to enable the predictive monitoring of a process, in addition to facilitate the discovery of the process model. 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 (e.g., Pareto distribution of traces, number of distinct variant-traces, number of distinct activities), in addition to the peculiarities of the process discovery algorithms. The identification of the guidelines for the choice of the preprocessing approach, as well as of the process discovery algorithm requires further investigation in future works.

Additional considerations concern the analysis of the time performance of the related methods. Fig. 10 reports the total computation times spent by FILTERING, SAMPLING, CLUSTERING and PROMISE$^+$ collected performing the process discovery with IMi. Similar conclusions can be drawn by using ILP or SM. These results show that PROMISE$^+$ is slower than SAMPLING. This is an expected result as SAMPLING does not perform any time-consuming machine learning stage for the event log summarization. Instead, PROMISE$^+$ is generally quicker than CLUSTERING that is the other sampling strategy that performs a machine learning stage. Exceptions are observed with Hospital, Road and Sepsis, which contain a smaller number of variants (compared to the remaining

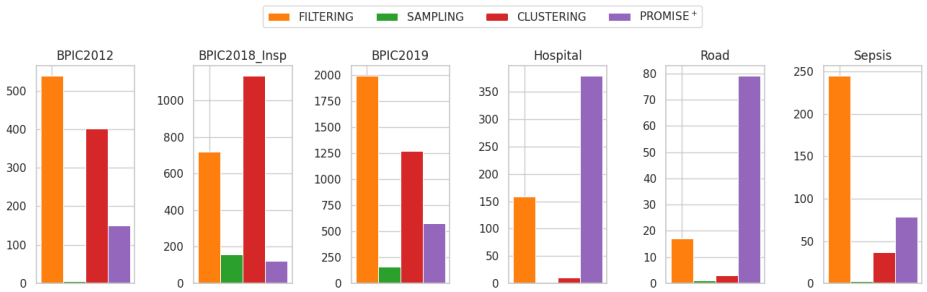
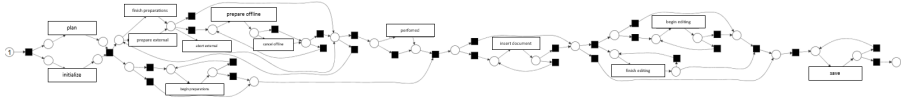


Figure 10: Total computation time spent in minutes by FILTERING, SAMPLING, CLUSTERING and PROMISE$^+$ approaches with Inductive Miner (IMi) as process discovery algorithm.

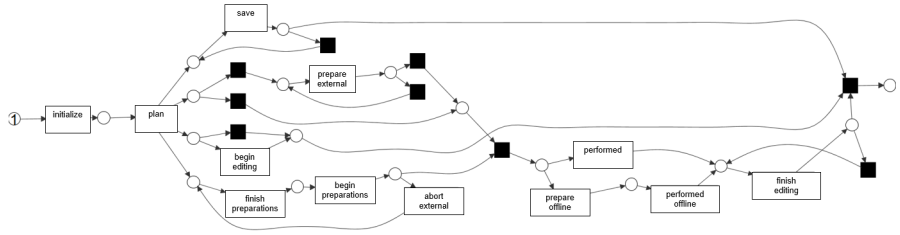
logs) to be explored through the time-consuming step of the clustering analysis. Finally, PROMISE$^+$ is, in general, quicker than FILTERING except for Hospital and Road, which are the logs that contain the smaller number of variants in this study. Surprisingly, the same behaviour is not observed in Sepsis that also contains a small number of variants. However, this event log contains a small amount of traces and events. The limited amount of data recorded in Sepsis allows us to complete the learning stage of both PROMISE$^+$ and CLUSTERING on a more borrowed time by making these methods competitive with FILTERING.

We complete this comparative study by analysing the Petri nets of the process models discovered with the methods: FILTERING, SAMPLING, CLUSTERING and PROMISE$^+$. As an example, let us consider the BPIC2018_Insp event log. Fig. 11a, 11b, 11c and 11d show the process models discovered with FILTERING, SAMPLING, CLUSTERING and PROMISE$^+$ from BPIC2018_Insp IMi. In accordance with the conclusions drawn by analyzing the complexity metrics measured with these methods, 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 Fmeasure 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 to the ability of capturing the behavior comprising the sub-sequence of activities “prepare external” and

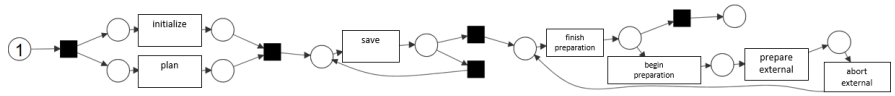


(a) FILTERING - precision=0.63, fitness=0.80, Fmeasure=0.70, model size=43×31×92, extended Cardoso index = 42.

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



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



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

Figure 11: Comparison of process models discovered by IMi with different sampling methods on BPIC2018_Insp.

"abort external". This subsequence also appears in the process model discovered with CLUSTERING (that achieves fitness equal to 0.96), but disappears in the process model discovered with SAMPLING. Finally, the process model discovered with PROMISE$^+$ provides an easy-to-interpret graphical explanation of the expected behaviour of the black-box LSTM model in predicting the next activity of a running trace. For example, it shows that "abort external" is expected in running traces ending with "begin preparation" and "prepare external".

7. 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 Fmeasure 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 Fmeasure 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. In particular, a future research direction involves the identification of guidelines for the choice of the preprocessing approach, as well as the process discovery algorithm.

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 this study, the greedy search is guided by the fitness and precision of the final process models discovered with the explored traces. However, the computation of fitness and precision is time consuming due to the time spent computing alignments. We plan to study alternative criteria 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 [15] 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.

8. 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., van Dongen, B., & van der Aalst, W. (2011). Conformance checking using cost-based fitness analysis. In *2011 IEEE 15th International Enterprise Distributed Object Computing Conference* (pp. 55–64).

[2] 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.

[3] 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.

[4] 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. In U. Dayal et al. (Ed.), *Business Process Management, 7th International Conference, BPM 2009, Proceedings* (pp. 159–175). Springer volume 5701 of *LNCS*.

[5] 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*, 1–39. doi:10.1142/S0218843014400012.

[6] 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,* (pp. 286–302). Springer volume 11675 of *LNCS*.

[7] 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.

[8] 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).

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

[10] van der Werf, 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.

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

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

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

[14] 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.

[15] 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).

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

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

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

[19] 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.

[20] Hanga, K. M., Kovalchuk, Y., & Gaber, M. M. (2020). A graph-based approach to interpreting recurrent neural networks in process mining. *IEEE Access*, *8*, 172923–172938.

[21] 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*.

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

[23] 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). Springer Berlin Heidelberg.

[24] 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*.

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

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

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

[28] 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.

[29] 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).

[30] 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*.

[31] 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)*, .

[32] 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*.

[33] 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.

[34] 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*.

[35] 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*.

[36] 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*.

[37] 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*.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] van Zelst, S., Bolt Iriondo, A., & van Dongen, B. (2017). Tuning alignment computation : an experimental evaluation. In W. van der Aalst, R. Bergenthum, & J. Carmona (Eds.), *Algorithms and Theories for the Analysis of Event Data 2017. Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data ATAED 2017* CEUR Workshop Proceedings (pp. 6–20).

[44] 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.

[45] Weijters, A., & Ribeiro, J. (2011). Flexible heuristics miner (FHM). *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)* IEEE Proceedings (pp. 310–317).

[46] 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.

[47] 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.

[48] 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.

[49] 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).