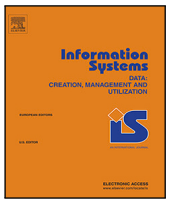
Information Systems 133 (2025) 102561

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



Stochastic conformance checking based on variable-length Markov chains

Emilio Incerto [a], Andrea Vandin [b,c,*], Sima Sarv Ahrabi [b]

[a] IMT Lucca, Piazza San Ponziano 6, Lucca, 55100, Italy
[b] L'EMbeDS and Institute of Economics, Sant'Anna School for Advanced Studies, Piazza Martiri della Libertà 33, Pisa, 56127, Italy
[c] DTU Technical University of Denmark, Anker Engelunds Vej 1, Bygning 101A, Kongens Lyngby, 2800, Denmark

ARTICLE INFO

Keywords:
Stochastic conformance checking
Earth Movers' Stochastic Conformance
Higher-order Markov chains

ABSTRACT

Conformance checking is central in process mining (PM). It studies deviations of logs from reference processes. Originally, the proposed approaches did not focus on stochastic aspects of the underlying process, and gave qualitative models as output. Recently, these have been extended in approaches for *stochastic conformance checking* (SCC), giving quantitative models as output. A different community, namely the *software performance engineering* (PE) one, interested in the synthesis of stochastic processes since decades, has developed independently techniques to synthesize Markov Chains (MC) that describe the stochastic process underlying program runs. However, these were never applied to SCC problems. We propose a novel approach to SCC based on PE results for the synthesis of stochastic processes. Thanks to a rich experimental evaluation, we show that it outperforms the state-of-the-art. In doing so, we further bridge PE and PM, fostering cross-fertilization. We use techniques for the synthesis of Variable-length MC (VLMC), higher-order MC able to compactly encode complex path dependencies in the control-flow. VLMCs are equipped with a notion of likelihood that a trace belongs to a model. We use it to perform SCC of a log against a model. We establish the degree of conformance by equipping VLMCs with uEMSC, a standard conformance measure in the SCC literature. We compare with 18 SCC techniques from the PM literature, using 11 benchmark datasets from the PM community. We outperform all approaches in 10 out of 11 datasets, i.e., we get uEMSC values closer to 1 for logs conforming to a model. Furthermore, we show that VLMC are efficient, as they handled all considered datasets in a few seconds.

1. Introduction

Process Mining (PM) is an interdisciplinary research area that aims at extracting insights and knowledge from execution traces of a process, bridging the gap between data science and process science [1]. PM involves a wide collection of techniques that can be grouped in three macro areas: process discovery, process enhancement, and conformance checking. Process discovery involves mining a graphical representation of the executed process, while process enhancement regards the enrichment of a model with additional information, such as the frequency of executed activities or paths. Conformance checking is a pivotal problem in PM, enabling the identification, analysis and fix of deviations among reference and mined processes [2]. Several proposals in PM focus on the stochastic nature of the studied process (see, e.g., [3–5], just to cite a few). However, historically, PM approaches, and in particular conformance checking ones, did not focus on the stochastic aspects of the underlying process, and considered qualitative models. That is, traditional conformance checking techniques have mostly ignored the stochastic perspective of the process model (see, e.g., the discussion in [6]). Recently, there has been a growing interest towards *stochastic conformance checking* (SCC, see, e.g., [7–9]), i.e., approaches to conformance checking that emphasize stochastic aspects of the underlying process, like the frequency and probability of traces, and consider quantitative models. The most recent among these approaches, therefore the state-of-the-art in stochastic conformance checking, are based of stochastic distances like the famous Earth Mover's Distance (EMD, also known as Wasserstein distance) [10], via SSC measures based on it named Earth Movers' Stochastic Conformance (EMSC), or unit EMSC (uEMSC) [7,9]. In these approaches, the reference model and a group of traces are transformed in two stochastic languages, respectively, and stochastic variants of the EMD distance among such languages are used to establish the conformance of the group of traces to the model.

Over the years, the so-called *software performance engineering* (PE) community, inherently interested in the synthesis and analysis of stochastic processes, developed techniques for synthesizing Markovian models that accurately describe the stochastic process underlying programs (see., e.g., [11–19]). However, surprisingly, stochastic conformance checking is not central in PE.

* Corresponding author.
E-mail addresses: emilio.incerto@imtlucca.it (E. Incerto), a.vandin@santannapisa.it (A. Vandin), s.sarvahrabi@santannapisa.it (S. Sarv Ahrabi).

https://doi.org/10.1016/j.is.2025.102561
Received 22 August 2024; Received in revised form 28 March 2025; Accepted 20 April 2025
Available online 9 May 2025

In this paper, we propose a novel approach to stochastic conformance checking based on PE techniques for the synthesis of Markovian models. Given in input a log, we show how to mine and give in output a Variable-length Markov Chain (VLMC, higher-order Markovian models equipped with memory) [11] from logs. Intuitively, VLMCs are Markov chains which partially depart from the usual *memory-less* property. Given a trace, they compute a trace-specific dependency on previous events, the memory, to compute a precise probability distribution for the next event. VLMCs are particularly well-suited for compactly expressing complex memory and path dependencies in the process. We use the VLMC notion of *likelihood* of a trace in a discovered stochastic process. The likelihood of a trace corresponds to the probability for the model to generate that trace. In line with the SCC literature (e.g., [20,21]), we use the notion of likelihood to perform SCC of a log against a model. In particular, we use the conformance measure uEMSC, standard in the SCC literature. We obtain an innovative method for stochastic conformance checking that is accurate. That is, our technique computes high uEMSC values, often close to 1, for logs conformant to models.

Our claims are supported by a rich experimental evaluation involving 11 benchmark datasets and 18 competitor SCC techniques from the PM literature. In particular, we align to the SCC literature in the sense that we use all datasets considered in [20], included in those in [21]. Furthermore, we compare with all the 15 SCC techniques considered in [20], and with the 3 additional ones considered in [21]. Notably, our approach outperforms all 18 competitor SCC techniques in terms of uEMSC values on 10 out of 11 datasets. That is, we get uEMSC values closer to 1. All experiments can be replicated using our replicability package [22]. Such good performances of our method may be due to the fact that all the considered competitor techniques are actually combinations of a qualitative discovery step, to mine the structure of a (qualitative) model, followed by a stochastic step where weights are assigned to the qualitative model to make it stochastic. Instead, our approach is natively stochastic: we directly learn a stochastic model (a higher-order Markov Chain). That is, the stochastic aspect is central also during the discovery of the structure. Another reason may be connected to the use of memory, which is central in the analysis of stochastic models in several domains (see, e.g. [23–25]). In fact, it allows to handle issues connected to the so-called phenomenon of path dependency [26–28]. As exemplified later in our running example in Listing 1, this may be an important dimension to consider. However, none of the considered competitor approaches is based explicitly on memory. In fact, our approach is not an improvement to an existing (PM) technique, but a new approach based on results coming from a different community. Nevertheless, our approach can be seen as 'in line' with the recent data-aware approach [20]. In fact, from an abstract point of view, our approach is based on a specific type of data: the 'memory' of each execution trace. In a later section, we also perform a preliminary study of the impact of noise in traces, showing a decrease in performance linear in the amount of noise in the dataset, and sketch an extension mitigating this effect.

2. Related work on stochastic conformance checking

As discussed in the Introduction, historically, conformance checking approaches focused on qualitative aspects of the underlying process. The obtained models were qualitative, ignoring the stochastic perspective of the underlying model. See, e.g., the discussion in [6,29]. More recently, there has been a growing interest towards SCC. That is, approaches to conformance checking that emphasize stochastic aspects of the underlying process, and produce stochastic models. See for example, just to cite a few [7–9]. Most of these techniques consider variants of the EMSC to establish the degree of conformance of a log with a model. Furthermore, in many cases the SCC techniques are obtained by first performing a qualitative discovery step which produces a qualitative model, followed by a quantitative step which makes the model stochastic. For example, the recent proposals in [20,21] share these characteristics. In our experiments in Section 5 we compare with the techniques proposed in the two papers, and with those included there for validation, for overall 18 SCC techniques. The authors of [20] propose a stochastic data-aware conformance checking technique which first mines a qualitative model, and then makes it stochastic by tuning weights in the model optimizing with respect to the impact of data variables. Instead, the authors of [21] propose a novel model based on variants of (generalized) stochastic Petri nets with silent transitions, and show how it can lead to improvements in the quality of existing SCC techniques by better estimating the likelihood (probability) of single traces in a model. Likewise, most of the approaches to SCC are based on (generalized) stochastic Petri nets, or fragments thereof [9,30–33]. We are not aware of SCC techniques based on higher-order Markov chains based on the use of memory.

A number of recent approaches to SCC tackle specific problems and scenarios. For example, the authors of [6] propose a flexible framework able to *partially match* traces, allowing to match traces which differ slightly. Another example is [34], which focuses on environments where event logs are assumed to be uncertain (known stochastically). Our take on SCC, the first one based on VLMCs and memory, may be extended in the future in these directions.

3. Performance engineering and VLMCs

In this section, we briefly introduce the domain of software performance engineering from a PM perspective (in Section 3.1), and then we treat in greater detail our reference model, VLMCs. In particular, we first introduce VLMCs informally based on an example (Section 3.2), and then introduce them formally (Section 3.3).

3.1. Overview on software performance engineering

Formal models of stochastic processes, such as Markov chains, play a critical role in the community of software performance engineering because they provide a mathematical framework for modeling and analyzing the probabilistic behavior of software systems [35,36]. These processes allow engineers to capture the randomness inherent in system performance, such as varying execution times, user interactions, and system workloads [37]. By representing a software system as a sequence of events with probabilistically determined transitions, Markov chains help in predicting system behavior under different conditions, identifying performance bottlenecks, and evaluating the impact of optimizations. This approach is essential for designing software that is not only efficient, but also resilient to the variability and uncertainty of real-world operating environments, enabling automated reasoning about a range of extra-functional, quantitative properties such as reliability, performance, and energy consumption [38,39].

Within PE, various techniques have been developed to automatically extract (i.e., to mine) Markov chains from executions of computer programs (i.e., from logs of program executions). As discussed, the aim of this paper is to show that techniques developed within the PE community can be applied to PM problems, and that this can advance the state of the art in PM. In particular, we show how leveraging techniques from PE can lead to accurate stochastic conformance checking. In Section 5 we demonstrate these improvements using well-established PM measures for stochastic conformance checking.

The key insight is to view a program as a low-level representation of the operational behavior of a business process. Using this interpretation, we can directly apply PE techniques to business process analysis, allowing to study the *processes underlying software programs* from the quantitative and stochastic point of view of PE.

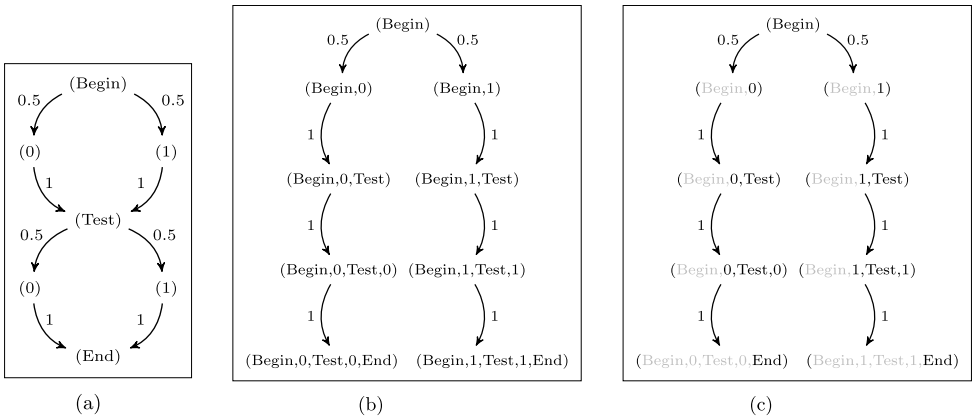


Fig. 1. Markov chains (MC) for Listing 1: (a) MC not encoding memory; (b) MC encoding full memory; (c) MC encoding relevant memory.

Listing 1: Python-like pseducode for a program with two highly inter-dependent if

```
1  prepareForNewTrace()
2  x = #Toss a fair coin: pick 0 or 1 with
       probability 0.5
3  addEvent("Begin")
4
5  if x == 0:
6      addEvent("0")
7  else:
8      addEvent("1")
9
10 addEvent("Test")
11 if x == 0:
12     addEvent("0")
13 else:
14     addEvent("1")
15
16 addEvent("End")
```

Table 1
The two possible traces/logs generated by Listing 1.

Case id	Timestamp	Activity	Case id	Timestamp	Activity
T0	1	Begin	T1	1	Begin
T0	2	0	T1	2	1
T0	3	Test	T1	3	Test
T0	4	0	T1	4	1
T0	5	End	T1	5	End

3.2. Overview on VLMCs

Running example. Let us consider the program in Listing 1. It logs a sequence of activities, i.e., {`Begin`, `Test`, `0`, `1`, `End`}, representing a simple process where the outcome of a fair coin toss (line 2) determines which events are recorded (lines 3, 6, 8, 10, 12, 14) followed by an `End` event (line 16). To track the behavior of the program in an event log, we included a logging infrastructure based on two functions: `prepareNewTrace` and `addEvent`. The former resets the events counter to zero, which is used to represent time, and generates a new `case ID` to group the events that belong to the same program execution. The latter, instead, increments the events counter and adds an entry to the event log for the activity name given as input, associating to it the latest generated `case ID` and `time`. Given that the random value is sampled once at the beginning of each execution (Line 2), it is easy to see that this program can generate only the two traces in Table 1.

Markov chains for the example in listing 1. By performing multiple statistically independent runs of a program, it is possible to collect event traces used to synthetize Markov chains representing the probabilistic behavior of the program [11,15–17,40,41]. For example, Fig. 1(a) illustrates a simple hand-crafted Markov chain which aims to represent the behavior of Listing 1. Each state corresponds to the emission of a specific activity whose probability is reported on the edge connecting two states. As it can be observed, the structure of this Markov chain closely follows the control-flow of the program. Indeed, in their simplest formulation, PE techniques for the synthesis of Markov chains underlying programs simply follow the branching structure of the program considering one instruction at a time, ignoring any dependence from previous instructions, that is, ignoring any dependency in the control-flow on history/memory [42]. This approach tends to produce compact models which, however, may not faithfully reflect the actual behavior of the program under analysis. Specifically, when replaying the generated model, it is possible to produce event traces that could actually not be generated by the original program. In process mining terms, this model has good fitness, but low precision. Considering the example in Listing 1, from both nodes `Begin` and `Test` of Fig. 1(a) we have two outgoing edges towards nodes `0` and `1` depending on the sampled value. However, this representation is accurate only for `Begin`. Instead, the chain is not able to express that when we get to `Test`, the next step is already pre-determined by the choice done in `Begin`: `Begin,0,Test` can only be followed by `0`, and `Begin,1,Test` by `1`.

In other words, in order to accurately describe only the admitted traces, it is necessary to encode a notion of *memory* within the chain to include the (hidden) non-local dependencies present in the control-flow (i.e., the actual business process logic). To address this issue, the main idea is to enrich the Markov chain inserting in its states the relevant memory. This leads to so-called higher-order Markov chains [43] which are then compiled back in (classic) Markov chains at the cost of a much larger state space. Fig. 1(b) illustrates informally a portion of the Markov chain underlying a higher-order Markov chain for the program in Listing 1. Each state has been expanded to encode the path followed so far (actually, the previous 5 steps). For example, state `Begin,0,Test` corresponds to state `Test` from Fig. 1(a) reached after executing `Begin` and `0`. This new Markov chain describes more accurately the program behavior, i.e., it expresses only the two traces in Table 1. However, it incurs in the so-called state space explosion problem. In fact, it is well-known that the Markov chain underlying a higher-order Markov chain with n states and memory k has n^k states [44]. Furthermore, it is complex, and therefore it fails to satisfy the *simplicity* criteria central in the PM community.

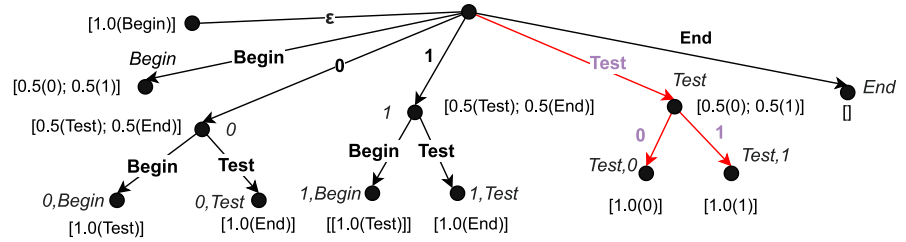


Fig. 2. PST modeling the behavior of program in Listing 1. Each edge is labeled with a symbol (an activity) emitted by the program while each node reports the probability distribution of the next symbol (within squared brackets). In addition, we mark each node with the juxtaposition of symbols encountered along the path from the root. The labels of root and leaves are omitted for space constraints. With ϵ we denote the empty trace. The PST is used to efficiently compute the probability distribution of the next symbol based on a suffix of the current trace (a path from root to a node). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Probabilistic-suffix tree for the example in listing 1. In order to address these dimensionality and complexity problems, approaches have been proposed in PE where an intermediate model represents implicitly and compactly the Markov chain underlying a higher-order Markov chain with memory. That is, the model does not explicitly enumerate all possible states, but only implicitly represents them. The state space can then be obtained on-the-fly by *executing*, or replaying, the model. This is akin, e.g., to stochastic Petri nets which generate large Markov chains based on an initial marking set. A well-known such approach, established within PE [11], Bioinformatics [45,46], NLP [47], and data compression [48], is based on *Variable-Length Markov Chains* (VLMCs) [23,49]. This is a formalism capable of compactly encoding a stochastic process with complex path dependencies in its control-flow by resorting to *variable-length memory*. Intuitively, e.g., in the example in Listing 1, the *relevant memory* is only the outcome of the first `if` statement, and this information is relevant only when executing the second `if` statement. This is intuitively depicted in the hand-made Markov chain in Fig. 1(c). Enriching states with only the relevant memory, typically leads to much fewer states than in the explicit Markov chain underlying the higher-order one [49].

As discussed, VLMCs are able to implicitly represent Markov chains with only relevant memory such as in Fig. 1(c). This is obtained by using a particular data structure to track which part of the process history is relevant in each state. This data structure, named *probabilistic suffix tree* (PST) [50], is used to determine the outgoing probability distribution of the next activity. The PST is built starting from the so-called *context function*, hereby denoted by $\mathcal{C}$. Let us consider a trace of activities $x_0x_1 \cdots x_n$, where x_0 is the initial activity, and x_n the final one.[1] The context of this trace, i.e., $\mathcal{C}(x_0x_1 \cdots x_n)$, is the shortest suffix $x_{n-k+1} \cdots x_n$ such that

$$\Pr(x_{n+1} \mid x_0x_1 \cdots x_n) = \Pr(x_{n+1} \mid x_{n-k+1} \cdots x_n)$$

for all possible one-step next activities x_{n+1}, where $\Pr(x_{n+1} \mid \cdots)$ is the probability of generating event x_{n+1} conditioned by the previous events in the trace. In other words, the context is the discussed *relevant memory*. The contexts of all states of a VLMC are compactly represented in the PST. In what follows, we informally discuss how to use PSTs and VLMCs on the example in Listing 1. We refer to Section 3.3 for more details and for the algorithms to synthesize them.

The PST of the program in Listing 1 is shown in Fig. 2 where with ϵ we denote the empty trace. Nodes are labeled (within squared brackets) with probability distributions of the next activities, conditioned on the relevant history (i.e., the labels of the edges from the node to the root). In addition, we mark each node with the juxtaposition of symbols encountered along the path from the root to that node. The PST is traversed to compute the probability distribution of the next activity. In particular, the traversal starts from the root, following the edge labeled with the latest produced activity. If the target node has an outgoing edge labeled with the second-latest produced activity, it means that more memory can be used to refine the probability distribution, and so on. Otherwise, we return the probability distribution labeling the current node. Let us consider the case in which the program has produced only the sub-trace `Begin`. That is, we compute

$$\{Pr(\mathtt{a} \mid \mathtt{Begin}),\ \mathtt{a} \in \mathcal{A}\},\ \text{with}\ \mathcal{A} = \{\mathtt{Begin}, 0, 1, \mathtt{Test}\}. \tag{1}$$

In the PST, we need to follow the top-left edge labeled with `Begin`. The target node has label `[0.5(0); 0.5(1)]`, implying that after the execution of `Begin`, the program will execute either 0 or 1 with probability 0.5. Let us consider, instead, the case in which the program has produced the sub-trace `Begin,0,Test`. That is, we compute

$$\{Pr(\mathtt{a} \mid \mathtt{Begin}, 0, \mathtt{Test}), \mathtt{a} \in \mathcal{A}\}. \tag{2}$$

We start by considering the latest activity, `Test`. That is, we consider the edge with different color having label `Test` in Fig. 2. The target node tells us that after the execution of this event, we have probability 0.5 for 0, and 0.5 for 1. This would be the result obtained without using any memory, akin to Fig. 1(a). However, given that the target node has outgoing edges, the PST tells us that this probability distribution can be refined by using one more step of memory. For doing so, we follow the edge labeled with 0, the activity preceding `Test` in the considered trace. We get to a state with probability 1 for event 0. In other words, given that two steps ago the program has executed 0, it can now only execute 0.

Given that no path in the PST has a length greater than two, this program can be described using memory of at most two preceding steps before generating a new one. PSTs enable variable-length memory, meaning that, as we have seen, some traces may require considering the last two activities, others only the last one, and in some cases, no previous activity at all.

Fig. 3. Fragments of the higher-order Markov chain with memory $m = 2$ corresponding to probability distribution of Eq. (1) and Eq. (2), respectively. In each fragment, a state is identified by a couple of random variables denoting the outcomes of the last two steps of the program reported in Listing 1. That is, each state only contains information about the last two generated symbols.

[1] In theory of VLMC and stochastic processes in general, traces (actually sequences) are usually denoted in reverse order, i.e., $x_nx_{n-1} \cdots x_0$. Here we avoid this for consistency with PM notation.

Markov chain of order m underlying the PST for the example in listing 1. A PST serves as an implicit representation of a higher-order Markov chain which uses memory of variable length. Representing a VLMC explicitly is complex and rarely done. Rather, PSTs are typically used to denote directly VLMCs, and therefore PSTs and VLMCs are often considered synonymous [23,49]. Instead, just for the sake of presentation, we can show a simpler, but usually much larger, (non higher-order) Markov chain which always uses the maximum memory supported by the PST (which, in our example is 2). In other words, thanks to the PST we know that we can construct a Markov chain with $|\mathcal{A}|^m$ states, with $\mathcal{A}$ the considered symbols, and m the maximum memory represented in the PST. In our example we would have 5^2 states, including the empty trace ϵ in the count of activities. Such Markov chain represents explicitly the probabilities that can be computed by the PST (probability of each next activity based on the last two activities), but incurs in the state-space explosion, and in fact it is neither constructed nor considered. We discuss it here just to provide intuition. Fig. 3a and b illustrate two fragments of this Markov chain, which can be used to derive the probability distributions in Eq. (1) and Eq. (2), respectively. For example, the root node in Fig. 3a represents the trace containing only `Begin`. Its outgoing edges state that the probability distribution for next activities assigns 0.5 to 0 and 1. Instead, the root node in Fig. 3b represents traces whose last two activities are `0,Test`, and states that for such traces, the probability distribution for next activities assigns 1 to 0, and 0 to any other symbol.

Trace likelihood in the VLMC for the example in listing 1. Here we move one step towards VLMC-based stochastic conformance checking by exemplifying how to use VLMCs and PSTs to compute the probability of a trace in a VLMC. We do this by introducing a notion of *likelihood* $\mathcal{L}$ for VLMCs. In the theory of stochastic processes, the likelihood of a trace (actually a sequence) of events (or symbols, or activities, depending on the domain) is a well-known notion [51]. It is the probability that a trace is generated by a stochastic process. E.g., it is used in maximum likelihood estimation [52], where one searches for the model that best fits the data (e.g., the model with highest average likelihood for all traces).

In words, the likelihood of a trace in a VLMC is the product of the probabilities of having each activity in the trace as next activity of the ones preceding it. Therefore, it is the probability of generating the trace in the considered model. Once the PST has been built, likelihoods can be easily calculated by adequately traversing the PST.

As a first example, we compute the likelihood of one of the two traces in Table 1 which, as discussed, belongs to the program in Listing 1 and to its PST in Fig. 2. In particular, we compute $\mathcal{L}(\texttt{Begin}, 0, \texttt{Test}, 0, \texttt{End})$. We divide the trace in five suffixes, for which we compute the following conditional probabilities according to what discussed in the previous paragraph:

$$Pr(\texttt{Begin}|C(\epsilon)) = Pr(\texttt{Begin}|\epsilon) = 1.0$$
$$Pr(0|C(\texttt{Begin})) = Pr(0|\texttt{Begin}) = 0.5$$
$$Pr(\texttt{Test}|C(\texttt{Begin}, 0)) = Pr(\texttt{Test}|\texttt{Begin}, 0) = 1.0$$
$$Pr(0|C(\texttt{Begin}, 0, \texttt{Test})) = Pr(0|0, \texttt{Test}) = 1.0$$
$$Pr(\texttt{End}|C(\texttt{Begin}, 0, \texttt{Test}, 0)) = Pr(\texttt{End}|\texttt{Test}, 0) = 1.0$$

By multiplying all such probabilities we compute 0.5 as the likelihood of the trace:

$$\mathcal{L}(\texttt{Begin}, 0, \texttt{Test}, 0, \texttt{End}) = 1.0 \cdot 0.5 \cdot 1.0 \cdot 1.0 \cdot 1.0 = 0.5 \quad (3)$$

As a second example, we compute $\mathcal{L}(\texttt{Begin}, 0, \texttt{Test}, 1, \texttt{End})$. That is, we compute the likelihood of a trace that does not belong to the program, nor to its PST. This time, we divide the trace in four suffixes, for which we compute the following conditional probabilities :

$$Pr(\texttt{Begin}|C(\epsilon)) = 1.0$$
$$Pr(0|C(\texttt{Begin})) = 0.5$$
$$Pr(\texttt{Test}|C(\texttt{Begin}, 0)) = 1.0$$
$$Pr(1|C(\texttt{Begin}, 0, \texttt{Test})) = 0.0$$

We have considered one suffix less than in the previous case because suffix `Begin,0,Test,1` does not belong to the PST. By multiplying such probabilities, we obtained 0 as likelihood:

$$\mathcal{L}(\texttt{Begin}, 0, \texttt{Test}, 1, \texttt{End}) = 1.0 \cdot 0.5 \cdot 1.0 \cdot 0.0 = 0.0$$

We can see the following: for the first trace, which belongs to the program, we got a non-zero likelihood (i.e., it has non-zero probability to be generated by the model). Instead, for the second trace, which does not belong to the program, we got 0 as likelihood (i.e., it has zero probability of being generated by the model).

3.3. Formal definition of VLMC

After the informal example-driven discussion given in Section 3.2, we now formally define VLMCs and their discovery algorithm originally introduced in [11,49].

As we have discussed, the relevant notions in VLMCs are context functions and PSTs (e.g., see Fig. 2), which we now formally introduce. As usual in the domain of stochastic processes [35], our definitions are based on an alphabet of symbols. In PM terms, these are the activities performed by the process.

Definition 1 (*Context*)**.** Let $\mathcal{A}$ be a finite alphabet, and let $\mathcal{A}^*$ denote the set of all finite traces over $\mathcal{A}$. A trace $v \in \mathcal{A}^*$ is called a *context* of a trace σ if v is a suffix of σ.

Definition 2 (*Context Function*)**.** Let $(X_t)_{t\in\mathbb{Z}}$ be a stochastic process taking values in $\mathcal{A}$. The *context function* is the mapping C: $\mathcal{A}^* \to \mathcal{A}^*$, such that for any trace σ, $C(\sigma)$ is the shortest suffix of σ such that

$$Pr(w \mid \sigma) = Pr(w \mid C(\sigma)).$$

for any $w \in A$. In other words, $C(\sigma)$ returns the minimal suffix of σ that contains all the necessary information to determine the future behavior of the process X.

Definition 3 (*Probabilistic Suffix Tree*)**.** Let $\mathcal{A}$ be a finite alphabet, and let $(X_t)_{t\in\mathbb{Z}}$ be a stochastic process taking values in $\mathcal{A}$, i.e., producing symbols in $\mathcal{A}$. Given a context function $C : \mathcal{A}^* \to \mathcal{A}^*$, a *Probabilistic Suffix Tree (PST)* is a tree T whose nodes are labeled by sequences from $\mathcal{A}^*$ that satisfies:

- The leaves of T correspond to the contexts defined by C, ensuring that for any trace $\sigma \in \mathcal{A}^*$, the unique minimal suffix $C(\sigma)$ is a leaf.
- If u is a child of v in T, then v is a suffix of u, forming a suffix-based hierarchical structure.
- Each node v is associated with a probability distribution $\{Pr(w \mid v) \mid w \in \mathcal{A}^*\}$, encoding the variable-length memory of the stochastic process.

For example, considering the PST in Fig. 2, the node `Test` with incoming red edge labeled `Test` has probability distribution assigning 0.5 to the activity 0 which leads to node `Test,0`, probability 0.5 to activity 1 which leads to node `Test,1`, and probability 0 to all other activities.

We can now formally introduce the notion of likelihood.

Definition 4 (*Trace Likelihood in a VLMC*)**.** Let $\mathcal{A}$ be a finite alphabet, and let $(X_t)_{t\in\mathbb{Z}}$ be a stochastic process taking values in $\mathcal{A}$. Given a trace $\sigma = a_0, a_1, \ldots, a_n \in \mathcal{A}^*$, and a PST T with context function $C : \mathcal{A}^* \to \mathcal{A}^*$, the likelihood $\mathcal{L}(\sigma)$ of observing σ is given by in T:

$$\mathcal{L}(\sigma) = Pr(a_0) \prod_{i=0}^{n-1} Pr(a_{i+1} \mid C(\sigma^i)). \quad (4)$$

where σ^l denotes the sub-trace of σ ending in position l.

Finally, we can discuss how VLMCs are mined. Here we use our algorithm from [11], adapted for PM notation. The algorithm is sketched in Algorithm 1 while we refer to [11] for a in-depth discussion.

First, *MineVLMC* sorts the events in the log in ascending order based on their timestamps (line 2) and constructs the suffix array [53] from the log L (line 3). This enables the efficient counting of occurrences for any given context and will be utilized in subsequent stages to compute all next-activity probability distributions efficiently.

For each activity a_i in L, the algorithm constructs a candidate sub-PST rooted at a_i, referred to as the *activity PST* (line 6). This sub-PST includes all suffixes in L that end in a_i, as they represent all possible memory dependencies for that activity. Additionally, the function *growPST* computes all next-activity probability distributions for the activity PST of a_i. Intuitively, it recursively builds a PST for activity a_i by expanding historical sequences backward. It initializes a node for the current context, calculates transition probabilities, and selectively extends the context if the new sequence is frequently observed. This ensures that only significant patterns contribute to the model, enhancing its efficiency in predicting future activities based on past ones. For further details on *growPST*, we refer to [11].

Finally, during the pruning phase, only statistically significant nodes are retained for each activity PST (line 7), which is then added to the final PST (line 8).

Algorithm 1: Mine a VLMC from a Event Log L

```
1  MineVLMC (L, n_min, α)
       inputs: The event log; The minimal occurrences of a
               context; The pruning factor
       output: A VLMC for L
2      L ← sort(L, Asc)
3      Global sa ← buildSuffixArray(L)
4      PST ← ∅
5      foreach Act a_i ∈ L do
6          PST_{a_i} ← growPST(a_i)
7          prune(PST_{a_i})
8          PST.addChild(PST_{a_i})
9      end
10     return PST
```

4. VLMC-based stochastic conformance checking

We are now in a position to propose a VLMC-based approach to stochastic conformance checking. In particular, we show how to incorporate the notion of trace likelihood computed in VLMCs in well-established metrics for stochastic conformance checking from PM.

Stochastic conformance checking evaluates how closely a stochastic process model aligns with an observed event log based on their probability distributions. Unlike classical conformance checking, which focuses on structural behavior, stochastic conformance checking compares the likelihood of observed traces with their corresponding probabilities in the model. This approach ensures that the model captures the possible behaviors, and also accurately reflects the observed frequencies or probabilities of those behaviors.

To assess the quality of a stochastic model, a common measure is the overlap in probability mass between the stochastic language of an event log and the stochastic language of the model. For stochastic process models, this overlap is often quantified by the *Unit Earth Movers' Stochastic Conformance* (uEMSC) measure [9]. The uEMSC calculates the overlap for each trace σ in the log L, measuring the positive difference between the probability of that trace in the log and its probability in the stochastic process model M [9]:

$$\text{uEMSC}(L, M) = 1 - \sum_{\sigma \in L} \max(L(\sigma) - M(\sigma), 0) \tag{5}$$

Table 2
Likelihood values for each trace σ generated by Listing 1. Column $M(\sigma)$ represents the likelihood computed from the mined PST, while column $L(\sigma)$ reports the likelihood measured from a log obtained by running 1 000 executions of Listing 1.

σ	$L(\sigma)$	$M(\sigma)$
`Begin, 0, Test,0,end`	0.5	0.5
`Begin, 1, Test,1,end`	0.5	0.5

Table 3
Details of the experimental set-up. Table (a) lists the used logs. These are all the logs in [20], taken from its replicability package [54]. These are also considered in [21]. Table (b) denotes the used stochastic conformance measure. Competitor stochastic conformance checking techniques have been obtained by combining the qualitative discovery techniques in Table (c) with the stochastic ones in Table (d). In both cases, these are all the techniques considered in [20,21].

(a) Logs

Log	Traces	Events	Activities
bpic12-a	6 562	30 541	10
bpic13-incidents	3 786	32 825	13
bpic13-open problems	412	1 179	5
bpic13-closed problems	748	3 361	7
bpic17-offer log	21 455	96 752	8
bpic20-domestic declarations	5 228	28 108	16
bpic20-international declaration	3 204	35 815	34
bpic20-prepaid travel cost	1 052	9 164	29
bpic20-request for payment	3 447	18 458	18
Sepsis	526	77 615	16
Road traffic fines	75 167	280 779	11

(b) Measure

uEMSC as defined by Eq. (5)

(c) Qualitative discovery techniques

Considered in [20,21]	
Directly Follows Model Miner	DFM
Considered in [20]	
Inductive Miner infrequent	IMF
Flower model: a model that allows for any behavior of the observed alphabet	FM

(d) Stochastic techniques

Considered in [20]	
Baseline uniform choices	BUC
Alignment-based estimator	ABE
Frequency-based estimator	FBE
Data-based stochastic discovery without one hot encoding	DSDwe [20]
Data-based stochastic discovery	DSD [20]
Considered in [21]	
Frequency-based estimator with enhanced likelihood estimation	FBEes [21]
Bill Clinton-based estimator with enhanced likelihood estimation	BCEes [21]
Alignment-based estimator with enhanced likelihood estimation	ABEes [21]

This formula relies on the probability of a trace σ in the log ($L(\sigma)$), which can be easily compute as the frequency of the trace in the log, and on its probability in a stochastic process model ($M(\sigma)$). The latter can be challenging to compute in general. Instead, in VLMCs it corresponds to the likelihood of the trace which can be easily computed using the PST.

As an example, we now compute uEMSC (Eq. (5)) for our running example in Listing 1. As discussed, the program can only produce the two traces in Table 1. We have run the program 1 000 times, obtaining 500 times each of the two traces. This is our language L. Running the process discovery algorithm discussed in Section 3.3, we obtain the PST in Fig. 2. This is our model M, and we set $M(\sigma) = \mathcal{L}(\sigma)$ for any trace σ. That is, we use as probability of a trace in a model its likelihood. As shown in Table 2, the likelihood of the two traces in Table 1 is 0.5. By doing the math, applying Eq. (5) to this data results in a uEMSC value of 1, indicating perfect conformance between the mined PST and the traces in the log.

5. Numerical evaluation

This section presents a comprehensive numerical evaluation of our VLMC-based approach for stochastic conformance checking comparing it with 18 stochastic conformance checking techniques on 11 benchmark datasets from the PM literature. All experiments can be replicated using our replicability package available on Zenodo [22].

5.1. Experimental setup

In this evaluation we take as reference two recent papers which present a novel stochastic conformance checking technique [20,21]. Specifically, precisely as done in the two reference papers, *(i)* we split a log under consideration in 2 parts containing 50% of the traces each, *(ii)* we mine a model (in our case a VLMC) using the traces in the first part, and *(iii)* perform stochastic conformance checking of the second part of the log against the mined model by computing the standard measure uEMSC (see Eq. (5)). In order to make the computed uEMSC values independent from the performed split, the experiment is repeated on 10 random splits of each log. Table 3 provides details about the experimental setup. Table 3(a) lists the datasets we consider. These are all the 11 popular PM benchmarks taken from [20], as available in its replicability package in Zenodo [54]. These are also considered in [21]. For each log, we use the 10 splits available in [54]. Table 3(b) provides the used stochastic conformance measure. Table 3(c)–(d) provide information on the considered competitor techniques in the literature. In particular, we use all the 15 and 3, respectively, stochastic conformance checking techniques considered in the two reference papers, for overall 18 techniques. All such 18 techniques actually consist of two steps: first a qualitative discovery step is performed, to mine the structure, i.e., a qualitative model. Then, a stochastic step makes the mined model stochastic, e.g., by assigning weights. Table 3(c) lists the 3 qualitative techniques used in the two reference papers, while Table 3(d) the 8 stochastic ones. The 15 techniques from [20] were obtained by combining all 3 qualitative techniques with the top 5 stochastic ones. For the sake of presentation, here we only report results on each stochastic technique, picking the qualitative technique where it performed best. Instead, we report all the 3 techniques from [21], which where obtained combining the qualitative technique DFM with the bottom 3 stochastic ones.

To assess the quality of our VLMC-based stochastic conformance checking, we compare the uEMSC values computed by it with the ones computed by all the other techniques, as provided in the corresponding papers (from the replicability package [54] of [20], and from Table A.2 of [21]).

5.2. Quality of the stochastic conformance checking task

The results of this evaluation are shown in Figs. 4 and 5. Each plot corresponds to one of the 11 logs in Table 3(a). For each log, we show the distribution of uEMSC values obtained for the 10 splits using each considered stochastic conformance technique. Each distribution is represented as a box spanning from the first quartile (25th percentile) to the third quartile (75th percentile), while the horizontal line within the box indicates the mean value. For the 3 techniques from [21], we only show the mean value, the only information available in [21]. The labels in the x-axis denote the used technique (`qualitative-discovery-technique_stochastic-technique`). For our VLMC-based we do not have this representation because, as discussed, our technique directly learns a stochastic model (i.e., it learns both the qualitative and stochastic aspects in a single step).

The results clearly demonstrate the high performance of VLMC-based stochastic conformance checking. In particular, we outperform all the other approaches in 10 out 11 datasets (~90%). That is, we achieve uEMSC values closer to 1, with cases where our uEMSC values are 3 times larger than all competitor methods (e.g., Fig. 4(j)), or 4 (e.g., Fig. 4(g)). Instead, there is not a clear winner among the other techniques.

We note that all the distributions of uEMSC values computed by our VLMC-based approach are compact, meaning that our approach does not seem to be particularly affected by specific splits. Instead, e.g., `DFM_DSD` creates a particularly spread distribution for one dataset (Fig. 4(c)).

Table 4
Runtime of our VLMC-based technique for all the datasets considered in Section 5.1. Each runtime is the mean for all splits. We can easily handle all datasets in seconds.

Log	*Runtime of VLMC-based method (s)*	
	Process discovery	*Stochastic conformance checking*
bpic12-a	9.50E−01	2.38E−03
bpic13-incidents	3.40E+00	2.67E−03
bpic13-open problems	3.72E−01	2.22E−03
bpic13-closed problems	4.36E−01	2.39E−03
bpic17-offer log	3.37E+00	2.51E−03
bpic20-domestic declarations	2.72E+00	2.46E−03
bpic20-international declaration	2.72E+00	2.51E−03
bpic20-prepaid travel cost	9.74E−01	2.36E−03
bpic20-request for payment	2.10E+00	2.35E−03
Sepsis	1.88E+00	2.43E−03
Road traffic fines	1.25E+01	2.31E−03

5.3. Runtime of the stochastic conformance checking task

We conclude the experimental section by discussing the runtime of our VLMC-based approach. The results are shown in Table 4. We use all datasets and splits from Section 5.1. All runtimes were computed as the average of computations on all 10 splits per dataset. We used a standard laptop machine (Mac M1 Pro).

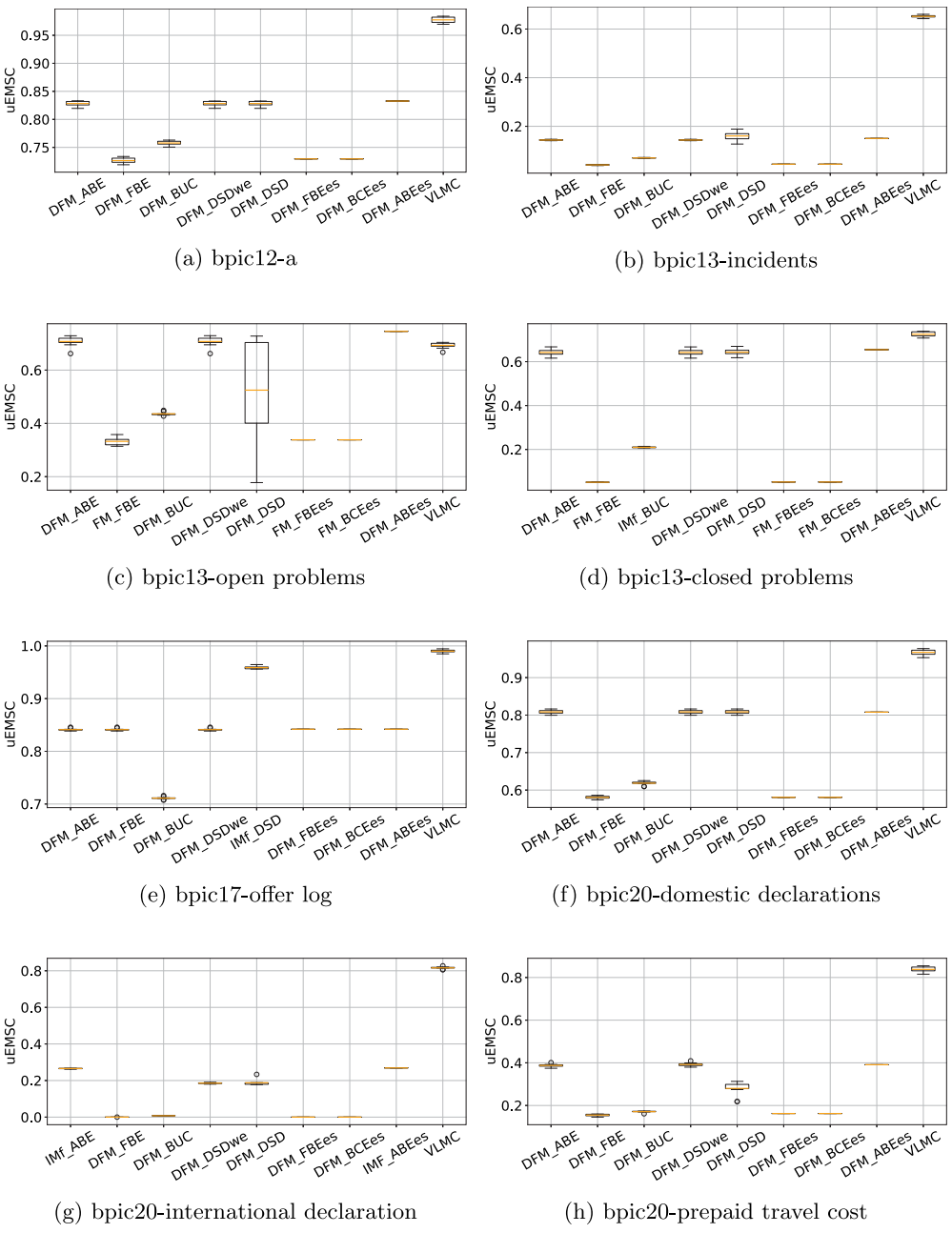
The table provides runtimes for process discovery, i.e., for learning the VLMC from each first part of a split, and for stochastic conformance checking, i.e., computing uEMSC values for each model using corresponding remaining part of the split. We note that our technique can easily handle all dataset in seconds. In particular, the process discovery took less than 1 s for 4 datasets, from 1 to 10 for 6 datasets, and only in one case it took more than 10 (12.5 s). Instead, stochastic conformance checking took in all cases only a few milliseconds. This is because VLMCs are particularly efficient in computing likelihoods, with guaranteed complexity logarithmic in the depth of the model [11].

6. On the impact of noise to VLMC-based stochastic conformance checking to noise

In this section, we study the performance of our approach in the presence of noisy data. We use one of the real-life event logs popular within PM used in Section 5; namely the well-known *road traffic fines* [55]. In Section 6.1 we briefly discuss the event log structure and an extension we do on it to simulate the presence of noise (we use two classic types of noise studied in the PM community), while in Section 6.3 we report the numerical evaluation of our approach under varying percentage of noise.

6.1. The dataset

The road traffic fines dataset [55] is a comprehensive collection that captures the various steps in the lifecycle of traffic fines, from issuance to resolution. This dataset is widely used to analyze and enhance the efficiency and effectiveness of traffic fine management systems and has been extensively utilized in the process mining community [56–64]. It contains 145 800 fine traces recorded between January 2000 and June 2012. Each trace documents transitions between events such as `Payment`, `Receive Result Appeal from Prefecture`, `Create Fine`, `Insert Fine Notification`, `Send Fine`, `Send for`



(a) bpic12-a (b) bpic13-incidents

(c) bpic13-open problems (d) bpic13-closed problems

(e) bpic17-offer log (f) bpic20-domestic declarations

(g) bpic20-international declaration (h) bpic20-prepaid travel cost

Fig. 4. Comparison of stochastic conformance checking results computed by state-of-the-art approaches from the literature, and by or VLMC-based one. We provide one plot per PM dataset in Table 3(a). Each plot shows the distributions of uEMSC values obtained over the 10 splits of the dataset for each technique. First 8 datasets out of 11.

`Credit Collection`, `Send Appeal to Prefecture`, `Insert Date Appeal to Prefecture`, `Appeal to Judge`, `Add Penalty`, and `Notify Result Appeal to Offender`.

Most traces are short, with an average of four events per trace. In 43% of the traces the process concludes after two events: the fine is paid (`Payment`) before the notification letter is sent out (`Send Fine`). However, 51% of the traces include five or more events, indicating more complex processes.

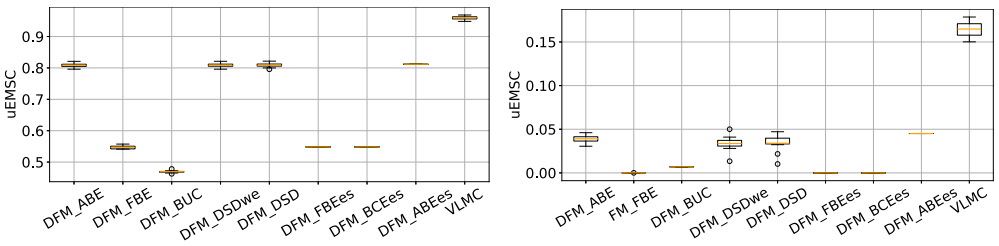
In order to use this dataset for stochastic conformance checking, we learn a VLMC on the original dataset, and we compute uEMSC on it for variations of the original log by introducing *random noise*. In particular, we considered a randomized version of *trace with additional event*, one of the *classic notions of noise in PM* described in Chapter 5.1.1 of [65]. In fact, for each trace we have chosen an event present in the trace, and we replicated it a random number of times from 1 to 3, adding the new events in random positions. We also considered a second type of classic notion of noise in PM: *missing episode of a trace* described in Chapter 5.1.1 of [65]. It is important to note that this method allowed us to generate non-trivial noise. The set of activities appearing in the original and modified traces is identical (for *trace with additional event*) or there is an inclusion relation (for *missing episode of a trace*), but they differ in their patterns. Both noise types are applied in controlled percentages to produce datasets with varying levels of data quality issues, allowing for a robust testing of our VLMC-based conformance checking.

6.2. Experiments

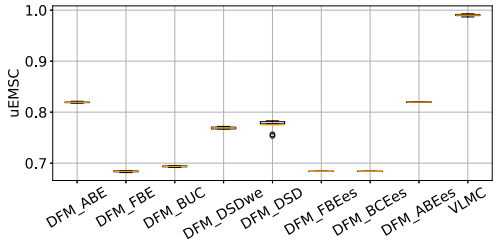
We now present the results obtained for the two types of noise.

6.2.1. Noise: trace with additional event

In order to apply this noise to a log we followed these steps:



(i) bpic20-request for payment (j) sepsis



(k) road traffic fines

Fig. 5. Comparison of stochastic conformance checking results computed by state-of-the-art approaches from the literature, and by or VLMC-based one. We provide one plot per PM dataset in Table 3(a). Each plot shows the distributions of uEMSC values obtained over the 10 splits of the dataset for each technique. Last 3 datasets out of 11.

Table 5
Impact of noise *trace with additional event* on uEMSC metric. The first column shows the percentage of traces affected by duplicative noise (5% to 40%), while the second column displays the corresponding uEMSC values.

%Noise	uEMSC
5	0.947
10	0.898
15	0.849
20	0.800
25	0.751
30	0.701
35	0.652
40	0.602

Table 6
Impact of noise *missing episode of a trace* on uEMSC metric. The first column shows the percentage of traces affected by removal noise (5% to 40%), while the second column displays the corresponding uEMSC values.

%Noise	uEMSC
5	0.949
10	0.901
15	0.854
20	0.805
25	0.757
30	0.710
35	0.662
40	0.614

- Select a random percentage of process instances (cases)
- For each selected case, randomly choose one event to duplicate
- Create from 1 to 3 duplicates of the chosen event, maintaining all original attributes
- Slightly adjust timestamps of duplicated events to preserve the chronological order

By following this procedure, we preserved the total number of unique cases while increasing the total event count. This noise simulates real-world data recording errors like system glitches, double-clicking, network issues causing retransmission, or redundant data entry. It increases the frequency of specific activities without altering the case structure or removing information.

Table 5 reports the uEMSC values computed by using our VLMC-based technique under an increasing percentage of noisy cases. The table demonstrates the impact of this type of noise on the uEMSC metric, showing a clear negative correlation between noise and conformance levels. As the percentage of noisy traces increases from 5% to 40%, the uEMSC score steadily decreases from 0.947 to 0.602, indicating that this type of noise linearly degrades the model's ability to accurately represent the underlying process.

6.2.2. Noise: missing episode of a trace

We apply this type of noise using steps similar to the ones from the previous section:

- Select a random percentage of process instances (cases)
- For each selected case, randomly remove from 1 to 3 events
- Ensure that each case retains at least one event after noise application

This type of noise simulates data loss where events fail to be recorded. For example, these could represent events that were executed but not captured in the information system due to manual recording errors, system crashes, or connectivity issues. It creates gaps in process execution traces that can significantly impact PM results.

Table 6 reports the uEMSC values computed using our VLMC-based technique under an increasing percentage of noisy cases. The table illustrates the effect of this type of noise on the uEMSC metric, revealing a consistent degradation in process model quality as noise increases. As the percentage of traces affected by event removal rises from 5% to 40%, the uEMSC score systematically decreases from 0.949 to 0.614, demonstrating how missing events linearly compromise the model's ability to accurately capture the true process behavior.

The two types of noise have a similar impact on VLMC-based stochastic conformance checking, with the latter leading to slightly better uEMSC values.

6.3. Towards VLMC-based SCC more flexible for noise

Our experiments suggest that VLMC-based SCC is sensible to noise in the data in a linear way. In future works, we plan to improve the *flexibility* of our approach towards noise. Intuitively, we have discussed

how to use VLMCs to compute the likelihood for a trace to be generated by them. This is computed iteratively by multiplying the likelihood of a number of sub-traces, where the number of necessary sub-traces varies trace by trace to handle complex dependencies on the memory of the model.

We have seen how this approach can produce very accurate results for noiseless data. However, it can be too restrictive in the presence of noise. A number of complementary proposals exist in the VLMC literature aiming at making VLMCs more tolerant to noisy data, even if not focusing on conformance checking. One common approach is the use of smoothing techniques, such as Laplace or Bayesian smoothing [66], which help assign a small probability to events that were not observed during training, thus potentially allowing to compute non-zero likelihood for traces altered by noise. Another possible strategy is to combine VLMCs with the so celled *skip-gram* method [67,68]. It extends the traditional VLMC framework by permitting gaps in the context (see Section 3.2), enabling the model to handle cases where noise introduces duplications or omissions of events (like the noise considered in the previous sections).

In what follows, we present an example sketching the use of skip-grams to compute the likelihood of a noisy version of the trace from Eq. (3). In particular, we duplicate the event `Test`:

$$\mathcal{L}(\texttt{Begin}, 0, \texttt{Test}, \texttt{Test}, 0, \texttt{End}) \tag{6}$$

We consider a VLMC model mined on noiseless data (see the PST of Fig. 2). Given that in the training data the context `Begin`, 0, `Test` was never followed by `Test`, we have

$$Pr(\texttt{Test} \mid \texttt{Begin}, 0, \texttt{Test}) = 0\,. \tag{7}$$

This leads to a likelihood of value 0 for the whole trace in Eq. (6).

In order to obtain a likelihood measure that is more robust to noise, we apply the so-called skip-gram method. This method includes a parameter named *skip*, which represents the number of activities in a noisy context that can be *skipped* to back off to a context observed during training. Intuitively, by *back off to a context* we mean that, when computing likelihoods, we replace the computation of probabilities of sub-traces such as the one in Eq. (7) by a weighted sum of probabilities including prefixes of the context. By setting a skip parameter of n, we are allowed to use prefixes of contexts with up to n fewer symbols. In this example, we set the skip parameter to 1, as it should be enough to ignore the duplicate event (i.e., the noise `Test`). By doing so, we back off to the context `Begin`, 0. Given that the training data contains traces where `Test` follows these two symbols, we expect to get non-zero likelihood. In particular, the computation of probability in Eq. (7) is replaced by the following weighted sum:

$$\begin{aligned} &Pr_{\text{skip}}(\texttt{Test} \mid \texttt{Begin}, 0, \texttt{Test}) \\ &\quad = \lambda\, Pr(\texttt{Test} \mid \texttt{Begin}, 0, \texttt{Test}) + (1-\lambda)\, Pr(\texttt{Test} \mid \texttt{Begin}, 0) \end{aligned}$$

where $0 \leq \lambda \leq 1$ is a hyperparameter that controls the relative weighting between the actual (in this case noisy) context, and the skip-gram one.

Choosing, for example, $\lambda = 0.7$ (i.e., assigning 70% weight to the direct context and 30% to the skip-gram context), we obtain:

$$Pr_{\text{skip}}(\texttt{Test} \mid \texttt{Test}, 0, \texttt{Begin}) = 0.7 \cdot 0 + 0.3 \cdot 1.0 = 0.3.$$

Thus, even though the VLMC trained on noiseless data never observed `Test` following `Begin`, 0, `Test` (i.e., $Pr(\texttt{Test} \mid \texttt{Begin}, 0, \texttt{Test}) = 0$), the skip-gram technique enables us to assign a non-zero probability to the noisy trace, compensating for the duplicated event.

7. Conclusion and future works

We introduced a novel approach to stochastic conformance checking (SCC) based on techniques from software performance engineering (PE) for the synthesis (i.e., mining) of Markovian processes from logs (program execution traces). In particular, we use state-of-the-art techniques from PE to learn higher-order Markovian models known as Variable-length Markov Chains (VLMC). Basing on an easy-to-compute and accurate notion of likelihood, we compare a log with a model by stochastic conformance checking. In particular, we use the standard SSC measure *unit Earth Movers' Stochastic Conformance* (uEMSC). Taking inspiration from the SCC literature [20,21], we perform a rich experimental comparison with 18 SCC competitor techniques using 11 popular benchmark datasets. We find that our approach outperforms the state-of-the-art in 10 out of 11 datasets. That is, we get *better* values of uEMSC (i.e., closer to 1). We also perform a preliminary study of the impact of noise in traces to our approach, and sketch an extension mitigating it.

Future works. A number of process mining techniques have been proposed particularly tailored for specific problems. For example, the approach in [6] allows to perform stochastic conformance checking ignoring small variations in traces by considering only partial matchings [6]. We plan to extend the *flexibility* of our approach towards partially matching traces, and evaluating how this may help also in handling incomplete [2] or noisy logs [65]. We have sketched an approach for this based on related works from the VLMC literature. This will lead to computing less precise but more permissive likelihoods. In process mining, approaches that do not present enough flexibility to handle the data at hand tend to produce very detailed and large models which overfit the data and are known as *spaghetti models*. A possible way to increase the flexibility of our approach in such cases could be investigating reduction and approximation techniques to simplify models as done in quantitative formal methods (e.g. [69–72]).

As we have seen, VLMCs can be compactly represented by probabilistic suffix trees (PST). There is a parallel among PSTs and procedural models like BPMN or Petri nets: all are compact representations able to represent large complex behavior. In this sense, a direction to pursue is evaluating and enhancing the *explainability* of our approach. This might involve studying automated analysis techniques for these formalisms like (statistical) model checking [73–75], or studying how informative for practitioners are PSTs compared to procedural languages, or inventing encodings of PSTs into procedural languages, thus *linking* our approach to information systems. Another direction to pursue is to investigate alignment procedures, a central feature of conformance checking, and techniques predictive monitoring using transition systems [76]. We are also interested in considering dynamic models which may evolve jointly with a *data* process. This may involve, e.g., defining dynamic variants of our context. In process mining, there may be an interest in representing forms of non-determinism in stochastic models. This is a limitation for VLMCs. In fact, VLMCs are (higher order) Markov chains, and therefore do not admit any non-determinism. In order to add non-determinism, one would have to move from Markov chains to Markov decision processes. We plan to investigate this extension.

modeloThis is our first attempt in narrowing the gap between the process mining and software performance engineering communities, possibly fostering cross-fertilization and opening new avenues for applications of stochastic conformance checking. For example, the high improved accuracy in stochastic conformance checking, based on tight likelihood estimations of single traces, may enable the stochastic extension of online conformance checking techniques (e.g., [77,78]), and enable tasks of multi-labeled classification: *to which stochastic process belongs a given trace?* (e.g., [79], where one of the results is that non-stochastic conformance checking alone is not enough to succeed in classification tasks, requiring to integrate it with machine learning approaches). We may also extend our approach towards the *time perspective*, by introducing notions of continuous-time VLMCs. Finally, we will investigate the integration of our approach with the ones based on product lines in [80,81], able to express complex constraints on data and resources.

CRediT authorship contribution statement

Emilio Incerto: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Andrea Vandin:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Sima Sarv Ahrabi:** Writing – original draft, Software, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank Andrea Burattin, DTU Denmark, for discussions and precious suggestions in several phases of the preparation of this work. The work has been partially supported by project SMaRT COnSTRUCT (CUP J53C24001460006), in the context of FAIR (PE0000013, CUP B53C22003630006) under the National Recovery and Resilience Plan (Mission 4, Component 2, Line of Investment 1.3) funded by the European Union - NextGenerationEU.

Data availability

We uploaded data and replicability material on zenodo. The link is in the paper.

References

[1] W.M.P. van der Aalst, Process Mining: Data Science in Action, second ed., Springer, Germany, 2016, http://dx.doi.org/10.1007/978-3-662-49851-4.

[2] J. Carmona, B. van Dongen, M. Weidlich, Conformance checking: Foundations, milestones and challenges, in: W.M.P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, Springer International Publishing, Cham, 2022, pp. 155–190, http://dx.doi.org/10.1007/978-3-031-08848-3_5.

[3] S.J.J. Leemans, T. Li, M. Montali, A. Polyvyanyy, Stochastic process discovery: Can it be done optimally? in: Advanced Information Systems Engineering: 36th International Conference, CAiSE 2024, Limassol, Cyprus, June 3–7, 2024, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2024, pp. 36–52, http://dx.doi.org/10.1007/978-3-031-61057-8_3.

[4] S.J.J. Leemans, F.M. Maggi, M. Montali, Reasoning on labelled Petri nets and their dynamics in a stochastic setting, in: Business Process Management: 20th International Conference, BPM 2022, MÜnster, Germany, September 11–16, 2022, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2022, pp. 324–342, http://dx.doi.org/10.1007/978-3-031-16103-2_22.

[5] A. Senderovich, M. Weidlich, L. Yedidsion, A. Gal, A. Mandelbaum, S. Kadish, C.A. Bunnell, Conformance checking and performance improvement in scheduled processes: A queueing-network perspective, Inf. Syst. 62 (2016) 185–206, http://dx.doi.org/10.1016/j.is.2016.01.002.

[6] E.G. Rocha, S.J.J. Leemans, W.M.P. van der Aalst, Stochastic conformance checking based on expected subtrace frequency, in: 2024 6th International Conference on Process Mining, ICPM, 2024, pp. 73–80, http://dx.doi.org/10.1109/ICPM63005.2024.10680673.

[7] S.J. Leemans, W.M. van der Aalst, T. Brockhoff, A. Polyvyanyy, Stochastic process mining: Earth movers' stochastic conformance, Inf. Syst. 102 (2021) 101724, http://dx.doi.org/10.1016/j.is.2021.101724.

[8] E.G. Rocha, S.J.J. Leemans, W.M. van der Aalst, Stochastic conformance checking based on expected SubTrace frequency, in: 2024 6th International Conference on Process Mining, ICPM, 2024.

[9] S.J.J. Leemans, A.F. Syring, W.M.P. van der Aalst, Earth movers' stochastic conformance checking, in: T. Hildebrandt, B.F. van Dongen, M. Röglinger, J. Mendling (Eds.), Business Process Management Forum, Springer International Publishing, Cham, 2019, pp. 127–143.

[10] L. Rüschendorf, The wasserstein distance and approximation theorems, Probab. Theory Related Fields 70 (1) (1985) 117–129.

[11] E. Incerto, A. Napolitano, M. Tribastone, Statistical learning of Markov chains of programs, in: 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2020, Nice, France, November 17-19, 2020, IEEE, USA, 2020, pp. 1–8, http://dx.doi.org/10.1109/MASCOTS50786.2020.9285947.

[12] V. Cortellessa, A. Di Marco, P. Inverardi, Model-Based Software Performance Analysis, Springer, Berlin, Heidelberg, 2011.

[13] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, Model-Based Performance Prediction in Software Development: A Survey 30(5), 295–310 (2004).

[14] T. Ohmann, M. Herzberg, S. Fiss, A. Halbert, M. Palyart, I. Beschastnikh, Y. Brun, Behavioral resource-aware model inference, 2014, pp. 19–30.

[15] C. Ghezzi, M. Pezzè, M. Sama, G. Tamburrelli, Mining behavior models from user-intensive web applications, in: Proceedings of the 36th International Conference on Software Engineering, 2014, pp. 277–287.

[16] G. Mazeroff, V. De, C. Jens, G. Michael, G. Thomason, Probabilistic trees and automata for application behavior modeling, in: 41st ACM Southeast Regional Conference Proceedings, 2003.

[17] G. Mazeroff, J. Gregor, M. Thomason, R. Ford, Probabilistic suffix models for API sequence analysis of windows xp applications, Pattern Recognit. 41 (1) (2008) 90–101.

[18] G. Garbi, E. Incerto, M. Tribastone, Learning queuing networks by recurrent neural networks, in: Proceedings of the ACM/SPEC International Conference on Performance Engineering, 2020, pp. 56–66.

[19] E. Incerto, A. Napolitano, M. Tribastone, Learning queuing networks via linear optimization, in: Proceedings of the ACM/SPEC International Conference on Performance Engineering, 2021, pp. 51–60.

[20] F. Mannhardt, S.J.J. Leemans, C.T. Schwanen, M. de Leoni, Modelling data-aware stochastic processes - discovery and conformance checking, in: L. Gomes, R. Lorenz (Eds.), Application and Theory of Petri Nets and Concurrency, Springer Nature Switzerland, Cham, 2023, pp. 77–98.

[21] S.J. Leemans, F.M. Maggi, M. Montali, Enjoy the silence: Analysis of stochastic Petri nets with silent transitions, Inf. Syst. 124 (2024) 102383, http://dx.doi.org/10.1016/j.is.2024.102383.

[22] E. Incerto, A. Vandin, S. Sarv Ahrabi, Stochastic conformance checking based on variable-length Markov chains: Reaplication package, 2024, http://dx.doi.org/10.5281/zenodo.13348234.

[23] D. Ron, Y. Singer, N. Tishby, The power of amnesia: Learning probabilistic automata with variable memory length, Mach. Learn. 25 (2–3) (1996) 117–149.

[24] A. Galves, E. Löcherbach, Infinite systems of interacting chains with memory of variable length—a stochastic model for biological neural nets, J. Stat. Phys. 151 (2013) 896–921.

[25] A.-L. Bianne-Bernard, F. Menasri, L. Likforman-Sulem, C. Mokbel, C. Kermorvant, Variable length and context-dependent HMM letter form models for arabic handwritten word recognition, in: Document Recognition and Retrieval XIX, Vol. 8297, SPIE, 2012, pp. 52–59.

[26] A. Choueiri, E. Santos, Discovery of path-attribute dependency in man- ufacturing environments: a process mining approach, J. Manuf. Syst. 61 (2010) 54–65.

[27] H. Sun, W. Liu, L. Qi, X. Ren, Y. Du, An algorithm for mining indirect dependencies from loop-choice-driven loop structure via petri nets, IEEE Trans. Syst. Man, Cybern.: Syst. 52 (9) (2021) 5411–5423.

[28] L. Wen, J. Wang, J. Sun, Detecting implicit dependencies between tasks from event logs, in: Frontiers of WWW Research and Development-APWeb 2006: 8th Asia-Pacific Web Conference, Harbin, China, January 16-18, 2006. Proceedings 8, Springer, 2006, pp. 591–603.

[29] S.J. Leemans, A. Polyvyanyy, Stochastic-aware precision and recall measures for conformance checking in process mining, Inf. Syst. 115 (2023) 102197, http://dx.doi.org/10.1016/j.is.2023.102197.

[30] A.T. Burke, S.J. Leemans, M.T. Wynn, W.M. van der Aalst, A.H. ter Hofstede, A chance for models to show their quality: Stochastic process model-log dimensions, Inf. Syst. 124 (2024) 102382, http://dx.doi.org/10.1016/j.is.2024.102382.

[31] G. Bergami, F.M. Maggi, M. Montali, R. Peñaloza, A tool for computing probabilistic trace alignments, in: S. Nurcan, A. Korthaus (Eds.), Intelligent Information Systems, Springer International Publishing, Cham, 2021, pp. 118–126.

[32] H. Alkhammash, A. Polyvyanyy, A. Moffat, L. García-Bañuelos, Entropic relevance: A mechanism for measuring stochastic process models discovered from event data, Inf. Syst. 107 (2022) 101922, http://dx.doi.org/10.1016/j.is.2021.101922.

[33] S.J.J. Leemans, A. Polyvyanyy, Stochastic-aware conformance checking: An entropy-based approach, in: S. Dustdar, E. Yu, C. Salinesi, D. Rieu, V. Pant (Eds.), Advanced Information Systems Engineering, Springer International Publishing, Cham, 2020, pp. 217–233.

[34] E. Bogdanov, I. Cohen, A. Gal, Conformance checking over stochastically known logs, in: C. Di Ciccio, R. Dijkman, A. del Río Ortega, S. Rinderle-Ma (Eds.), Business Process Management Forum, Springer International Publishing, Cham, 2022, pp. 105–119.

[35] G. Bolch, S. Greiner, H. De Meer, K.S. Trivedi, Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, John Wiley & Sons, Hoboken, NJ, 2006.

[36] F. Randone, L. Bortolussi, E. Incerto, M. Tribastone, Inference of probabilistic programs with moment-matching gaussian mixtures, Proc. ACM Program. Lang. 8 (POPL) (2024) 1882–1912.

[37] G. Garbi, E. Incerto, M. Tribastone, μP: A development framework for predicting performance of microservices by design, in: 2023 IEEE 16th International Conference on Cloud Computing, CLOUD, IEEE, 2023, pp. 178–188.

[38] V. Stoico, V. Cortellessa, I. Malavolta, D. Di Pompeo, L. Pomante, P. Lago, An approach using performance models for supporting energy analysis of software systems, in: European Workshop on Performance Engineering, Springer, 2023, pp. 249–263.

[39] E. Incerto, R. Pizziol, M. Tribastone, μOpt: An efficient optimal autoscaler for microservice applications, in: 2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS, IEEE, 2023, pp. 67–76.

[40] A. Hindle, E.T. Barr, Z. Su, M. Gabel, P. Devanbu, On the naturalness of software, in: 2012 34th International Conference on Software Engineering, ICSE, IEEE, 2012, pp. 837–847.

[41] S. Wang, D. Chollak, D. Movshovitz-Attias, L. Tan, Bugram: bug detection with n-gram language models, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, 2016, pp. 708–719.

[42] R.C. Carrasco, J. Oncina, Learning stochastic regular grammars by means of a state merging method, in: International Colloquium on Grammatical Inference, Springer, 1994, pp. 139–152.

[43] W.K. Ching, E.S. Fung, M.K. Ng, Higher-order Markov chain models for categorical data sequences, Naval Res. Logist. 51 (4) (2004) 557–574.

[44] E. Ever, O. Gemikonakli, A. Kocyigit, E. Gemikonakli, A hybrid approach to minimize state space explosion problem for the solution of two stage tandem queues, J. Netw. Comput. Appl. 36 (2) (2013) 908–926.

[45] D. Dalevi, D. Dubhashi, M. Hermansson, A new order estimator for fixed and variable length Markov models with applications to DNA sequence similarity, Stat. Appl. Genet. Mol. Biol. 5 (1) (2006).

[46] F. Cunial, J. Alanko, D. Belazzougui, A framework for space-efficient variable-order Markov models, Bioinform.tics 35 (22) (2019) 4607–4616.

[47] M.J. Zaki, C.D. Carothers, B.K. Szymanski, Vogue: A variable order hidden markov model with duration based on frequent sequence mining, ACM Trans. Knowl. Discov. from Data (TKDD) 4 (1) (2010) 1–31.

[48] J.L. Nunez-Yanez, V.A. Chouliaras, A configurable statistical lossless compression core based on variable order Markov modeling and arithmetic coding, IEEE Trans. Comput. 54 (11) (2005) 1345–1359.

[49] P. Bühlmann, A.J. Wyner, et al., Variable length Markov chains, Ann. Statist. 27 (2) (1999) 480–513.

[50] G. Bejerano, G. Yona, Variations on probabilistic suffix trees: statistical modeling and prediction of protein families, Bioinform. 17 (1) (2001) 23–43.

[51] C.A. Rohde, et al., Introductory Statistical Inference with the Likelihood Function, Springer, Berlin, Heidelberg, 2014.

[52] I.J. Myung, Tutorial on maximum likelihood estimation, J. Math. Psych. 47 (1) (2003) 90–100.

[53] J. Lin, D. Adjeroh, B.-H. Jiang, Probabilistic suffix array: efficient modeling and prediction of protein families, Bioinform. 28 (10) (2012) 1314–1323.

[54] F. Mannhardt, S.J. Leemans, C.T. Schwanen, M. de Leoni, Supplementary material to modelling data-aware stochastic processes - discovery and conformance checking, 2023, http://dx.doi.org/10.5281/zenodo.7578655.

[55] M. De Leoni, F. Mannhardt, Road traffic fine management process, Eindh. Univ. Technol. Dataset 284 (2015).

[56] A. Burattin, F.M. Maggi, A. Sperduti, Conformance checking based on multi-perspective declarative process models, in: International Conference on Business Process Management, Springer, 2012, pp. 109–121.

[57] A. Burattin, F.M. Maggi, A. Sperduti, W.M. van der Aalst, Balanced multi-perspective checking of process conformance, in: International Conference on Business Process Management, Springer, 2014, pp. 179–190.

[58] A. Burattin, W.M. van der Aalst, An alignment-based multi-perspective online conformance checking technique, in: International Conference on Business Process Management, Springer, 2015, pp. 107–119.

[59] E.M. de Murillas, F.M. Maggi, CoCoMoT: Conformance checking of multi-perspective processes via smt (extended version), in: International Conference on Software Engineering and Formal Methods, Springer, 2019, pp. 220–236.

[60] A. Burattin, J. Carmona, W.M. van der Aalst, A multi-perspective online conformance checking technique, in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, ACM, 2014, pp. 813–820.

[61] F. Taymouri, M. La Rosa, F.M. Maggi, Efficient process conformance checking on the basis of uncertain event-to-activity mappings, in: International Conference on Advanced Information Systems Engineering, Springer, 2021, pp. 239–256.

[62] A. Burattin, J. Carmona, F.M. Maggi, W.M. van der Aalst, Towards multi-perspective conformance checking with aggregation operations, in: International Conference on Business Process Management, Springer, 2016, pp. 171–187.

[63] W.M. van der Aalst, F.M. Maggi, Conformance checking of processes based on monitoring real behavior, in: International Conference on Business Process Management, Springer, 2014, pp. 61–78.

[64] L. Wen, F.M. Maggi, S. Zugal, M. Montali, M. Song, W.M. van der Aalst, Temporal conformance checking at runtime based on time-infused process models, in: International Conference on Conceptual Modeling, Springer, 2018, pp. 364–379.

[65] C.C. Günther, Process mining in flexible environments, in: Technische Universiteit Eindhoven, (Ph.D. Thesis), 2009, p. 82, https://pure.tue.nl/ws/portalfiles/portal/3032467/200911996.pdf.

[66] S.F. Chen, J. Goodman, An empirical study of smoothing techniques for language modeling, in: Proceedings of the 34th Annual Meeting on Association for Computational Linguistics, 1996, pp. 310–318.

[67] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: Proceedings of the International Conference on Learning Representations, ICLR, 2013.

[68] S.A. Islam, B.J. Heil, C.M. Kearney, E.J. Baker, Protein classification using modified n-grams and skip-grams, Bioinform. 34 (9) (2018) 1481–1487.

[69] G. Bacci, G. Bacci, K.G. Larsen, G. Squillace, M. Tribastone, M. Tschaikowski, A. Vandin, Dissimilarity for linear dynamical systems, in: J. Hillston, S. Soudjani, M. Waga (Eds.), Quantitative Evaluation of Systems and Formal Modeling and Analysis of Timed Systems - First International Joint Conference, QEST+FORMATS 2024, Calgary, AB, Canada, September 9-13, 2024, Proceedings, in: Lecture Notes in Computer Science, vol. 14996, Springer, 2024, pp. 125–142, http://dx.doi.org/10.1007/978-3-031-68416-6_8.

[70] L. Cardelli, R. Grosu, K.G. Larsen, M. Tribastone, M. Tschaikowski, A. Vandin, Algorithmic minimization of uncertain continuous-time Markov chains, IEEE Trans. Autom. Control. 68 (11) (2023) 6557–6572, http://dx.doi.org/10.1109/TAC.2023.3244093.

[71] G. Argyris, A. Lluch-Lafuente, A. Leguizamon-Robayo, M. Tribastone, M. Tschaikowski, A. Vandin, Minimization of dynamical systems over monoids, in: 38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023, IEEE, 2023, pp. 1–14, http://dx.doi.org/10.1109/LICS56636.2023.10175697.

[72] D. Toller, M. Tribastone, M. Tschaikowski, A. Vandin, Coarse-graining complex networks for control equivalence, IEEE Trans. Autom. Control. 70 (2) (2025) 1169–1175, http://dx.doi.org/10.1109/TAC.2024.3448240.

[73] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, A. Vandin, A formal approach for the analysis of BPMN collaboration models, J. Syst. Softw. 180 (2021) 111007, http://dx.doi.org/10.1016/J.JSS.2021.111007.

[74] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, A. Vandin, Bprove: a formal verification framework for business process models, in: G. Rosu, M.D. Penta, T.N. Nguyen (Eds.), Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017, IEEE Computer Society, 2017, pp. 217–228, http://dx.doi.org/10.1109/ASE.2017.8115635.

[75] R. Casaluce, A. Burattin, F. Chiaromonte, A. Lluch-Lafuente, A. Vandin, White-box validation of quantitative product lines by statistical model checking and process mining, J. Syst. Softw. 210 (2024) 111983, http://dx.doi.org/10.1016/J.JSS.2024.111983.

[76] W. van der Aalst, M. Schonenberg, M. Song, Time prediction based on process mining, Inf. Syst. 36 (2) (2011) 450–475, http://dx.doi.org/10.1016/j.is.2010.09.001, Special Issue: Semantic Integration of Data, Multimedia, and Services.

[77] W.L.J. Lee, A. Burattin, J. Munoz-Gama, M. Sepúlveda, Orientation and conformance: A HMM-based approach to online conformance checking, Inf. Syst. 102 (2021) 101674, http://dx.doi.org/10.1016/J.IS.2020.101674.

[78] A. Burattin, S.J. van Zelst, A. Armas-Cervantes, B.F. van Dongen, J. Carmona, Online conformance checking using behavioural patterns, in: M. Weske, M. Montali, I. Weber, J. vom Brocke (Eds.), Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings, in: LNCS, 11080, Springer, Germany, 2018, pp. 250–267, http://dx.doi.org/10.1007/978-3-319-98648-7_15.

[79] M.F. Sani, F. Nikraftar, M. Sroka, A. Burattin, Behavioral recommender system for process automation steps, in: 12th International Conference on Data Science, Technology and Applications, 2023, pp. 255–262.

[80] M.H. ter Beek, A. Legay, A. Lluch-Lafuente, A. Vandin, A framework for quantitative modeling and analysis of highly (re)configurable systems, IEEE Trans. Softw. Eng. 46 (3) (2020) 321–345, http://dx.doi.org/10.1109/TSE.2018.2853726.

[81] M. Kowal, M. Tschaikowski, M. Tribastone, I. Schaefer, Scaling size and parameter spaces in variability-aware software performance models (t), in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE, 2015, pp. 407–417, http://dx.doi.org/10.1109/ASE.2015.16.