Stochastic Conformance Checking based on Variable-length Markov Chains

Emilio Incerto[1], Andrea Vandin[2*], Sima Sarv Ahrabi[2]

[1]IMT Lucca, Piazza San Ponziano 6, Lucca, 55100, Italy.
[2*]L'EMbeDS and Institute of Economics, Sant'Anna School for Advanced Studies, Piazza Martiri della Libertà 33, Pisa, 56127, Italy.

*Corresponding author(s). E-mail(s): a.vandin@santannapisa.it;
Contributing authors: emilio.incerto@imtlucca.it;
s.sarvahrabi@santannapisa.it;

Abstract

Conformance checking is central in process mining (PM). It studies deviations from reference processes. Historically, the focus was on qualitative aspects, rather than on quantitative ones. Recently, however, approaches for *stochastic conformance checking* emerged, with state-of-the-art ones based on stochastic distances like the Earth Mover's Distance (EMD). The *software performance engineering* (PE) community developed techniques to synthesize Markov Chains (MC) that describe the stochastic process underlying programs. We propose a novel, PE-inspired, approach to stochastic conformance checking. We aim at bridging PE and PM, fostering cross-fertilization. This may result, e.g., in importing in PM verification, control, and repair techniques popular in PE. We base on techniques for the synthesis of Variable-length MC (VLMC), higher-order MC able to compactly encode complex dependencies in the control-flow. We equip VLMCs with a notion of *likelihood* to efficiently and reliably study the conformance of traces from a control-flow perspective. We compare with EMD using a novel dataset from PE containing traces of two implementations of the sorting algorithmic task, as well as a classic PM dataset. We outperform EMD in runtime and accuracy, while we have limited flexibility for incomplete logs. We find that EMD is very sensible to the used *threshold* distance beyond which traces are marked non-conformant, while our approach is threshold-free, avoiding this critical requirement. We can check the conformance of single traces, while EMD performs best for groups of traces. This paves the way for future applications of stochastic conformance checking, e.g., in online PM, or in process-oriented classification tasks.

Keywords: Stochastic conformance checking, Earth Mover's Distance, Higher-order Markov chain, Software Performance Engineering

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]. Historically, approaches to conformance checking have emphasized qualitative aspects. Recently, there has been a growing interest towards *stochastic conformance checking* (see, e.g., [3–5]), i.e., approaches to conformance checking that emphasize quantitative aspects like the frequency and probability of traces. 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) [6]. In these approaches, the reference model and a group of traces are transformed in two stochastic languages, respectively, and the EMD distance among such languages is 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 quantitative aspects, developed techniques for synthesizing Markovian models that accurately describe the stochastic process underlying programs (see., e.g., [7–15]). However, surprisingly, stochastic conformance checking is not central in PE. In this paper, we propose a novel approach to stochastic conformance checking inspired by PE results. We show how to synthesize Variable-length Markov Chains (VLMC, higher-order Markovian models equipped with memory) [7] from program execution traces as well as from event logs in general. VLMCs are particularly well-suited for compactly expressing complex memory and path dependencies in the process. We equip VLMCs with a notion of *likelihood* that a trace conforms to the mined stochastic process. The likelihood of a trace corresponds to the probability for the model to generate that trace. Therefore, it is greater than 0 if the trace belongs to the language of the model (i.e., if it is conformant from the control-flow perspective), and 0 otherwise. We obtain an innovative method for stochastic conformance checking that is efficient and accurate. We identify two major differences among our approach and EMD-based ones: (i) EMD requires the choice of a *threshold* beyond which traces are considered non conformant, while our approach is threshold-free; (ii) our approach allows to check the conformance of single traces, while EMD-based ones work best when more traces are considered at once. We show that these differences are crucial, because EMD appears to be sensitive to the choice of threshold, and to have low performance in detecting the conformance of single traces.

Our claims are supported by an experimental evaluation involving two datasets: (i) a new dataset coming from the PE field containing traces of different implementations of an algorithmic task, sorting; (ii) a dataset on Italian vehicle fines popular within the PM community [16]. Using these datasets, we compare several aspects of VLMC- and EMD-based methods. Our approach outperforms the other in terms of runtime and

accuracy in all tested settings, but it may be limited in terms of flexibility to handle incomplete logs (the accuracy of VLMC in correctly identifying conformant traces appears to be closely correlated to the completeness of the logs). Notably, dataset (i), which we make available to the community at [17] together with replicability material, is one of the contributions of this paper.

This work aims to bridge the PM and PE communities, fostering cross-disciplinary proposals. It paves the way to a number of novel applications of stochastic conformance checking. For example, the improved performances for single traces may enable the stochastic extension of online conformance checking techniques (e.g., [18, 19]), and of multi-labeled classification tasks: *to which stochastic process belongs a given trace?* (e.g., [20], 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). Even though the focus of this paper is on stochastic conformance checking, we remark an additional benefit in bridging the two: once an accurate stochastic process is mined from execution traces, it is possible to verify that the underlying process satisfies required performance measures (e.g., [7]).

2 On stochastic conformance checking in process mining

Traditional approaches to conformance checking had emphasis on qualitative aspects, while, more recently, stochastic extensions focused on quantitative aspects emerged. The paper by Bogdanov et al. [21] focuses on environments where event logs are assumed to be uncertain (known stochastically). They develop an algorithm incorporating a cost function reflecting event uncertainty, enabling optimal alignment between a process model and stochastic event observations, improving the reliability in handling uncertain data. Further exploration considered stochastic extensions of models popular in the community, such as Petri nets. An approach based on stochastic Petri nets, exploiting silent transitions, is detailed in [22]. The considered formalism offers a reliable way to forecast process behavior and improve stochastic conformance checking by combining automata-based methods with absorbing Markov chains. The introduction of k^{th}-order Markovian abstractions [23] allows for a scalable evaluation of complex stochastic variations in process models in the presence of large datasets. Another development is the application of entropy measures to assess the recall and precision of process models and event logs in comparison to stochastic automata [24]. Ongoing efforts towards stochastic notions of conformance checking are further demonstrated by the development of a framework for estimating weights in Generalized Stochastic Petri Nets (GSPNs) [25] and the adaptation of process discovery algorithms to manage infrequent behaviors in event logs [26]. Research into the effectiveness of business process simulation models [27] and innovative approaches for the concept of drift detection [28] emphasize the need for dynamic adaptation and integrating time attributes in concept drift frameworks.

There exists a recent line of research which uses the Earth Mover's Distance [6] to perform stochastic conformance checking. EMD is a notion of distance among stochastic languages to assess the conformance between stochastic process models and event

logs. For example, the work by Leemans et al. [3] introduces a method based on EMD where both the process model and a group of event logs are mapped onto stochastic languages that are then compared in terms of EMD distance. The method takes into account event frequencies and their probabilities, and uses a reallocation matrix and stochastic trace alignments to identify discrepancies and align observed behaviours with modelled probabilities. Another example is the Toothpaste Miner [29]. It mines stochastic Petri nets, and applies EMD not on the mined process, but on traces simulated by it. Another recent approach based on EMD is [4]. It presents a flexible framework for stochastic conformance checking supporting partially matching traces.

The capabilities of process mining have been greatly expanded by developments tailored to stochastic conformance checking. However, integrating stochastic information in the picture is still a challenge, particularly when it comes to controlling complexity and scalability, as well as the ability in practice in detecting conformant traces. In this paper, we present our own contribution in this line of research, aiming at addressing these open issues. Given that the several recent approaches to stochastic conformance checking are connected to EMD, we consider EMD-based approaches state-of-the-art, and present a thorough comparison with EMD only.

3 Performance engineering meets stochastic conformance checking

In this section, we review approaches from the software performance engineering community for the synthesis of stochastic processes (Sections 3.1-3.2), and propose a novel approach for stochastic conformance checking based on such results (Section 3.3).

3.1 Overview

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 [30]. These processes allow engineers to capture the randomness inherent in system performance, such as varying execution times, user interactions, and system workloads [31]. 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 [32, 33].

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 bridge PM and PE by demonstrating that techniques from the latter can be effectively tailored to central tasks of PM. In particular, focusing on stochastic conformance checking, we show how leveraging techniques from PE can enhance PM towards more accurate and efficient stochastic process-oriented analyses. To unite these two domains, the key insight is to

Listing 1: Python-like pseudocode 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")
```

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

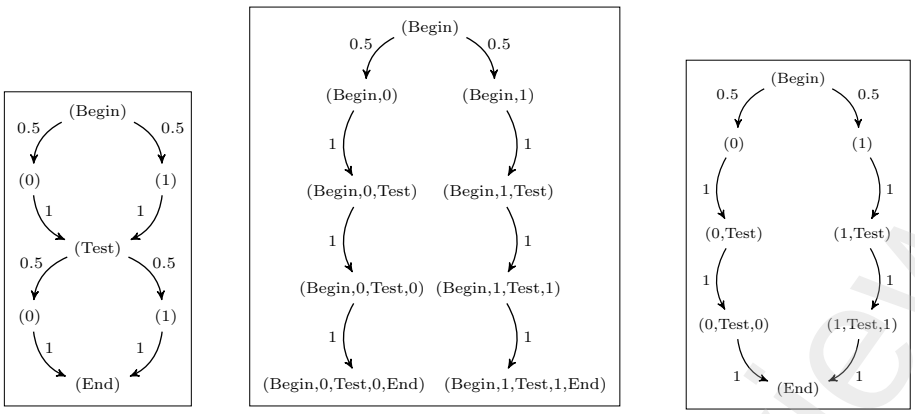
Case id	*Timestamp*	*Activity*
T1	1	Begin
T1	2	1
T1	3	Test
T1	4	1
T1	5	End

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

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.

3.2 Synthesis of stochastic processes

Let us consider the program in Listing 1. It logs a sequence of events, 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.



(a) (b) (c)

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

Synthesis of Markov chains

By executing programs across multiple statistically independent runs, it is possible to collect event traces used to synthetize Markov chains representing the probabilistic behavior of the program [7, 11–13, 34, 35]. For example, Figure 1a illustrates a simple Markov chain that represents the behavior of Listing 1, where each state corresponds to the emission of a specific event 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. This approach tends to produce compact models which, however, may not faithfully reflect the actual behavior of the program under analysis. Specifically, when simulating the generated model, it is possible to produce event traces that could actually not be generated by the original program. Considering the example in Listing 1, from both nodes `Begin` and `Test` of Figure 1a 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) constraints 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 [36] which are then compiled back in (classic) Markov chains, at the cost of a much larger state space. Figure 1b illustrates a Markov chain where each state has been expanded to encode the path followed so far: state `Begin,0,Test` corresponds to state `Test` from

Figure 1a 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 [37]

Synthesis of Variable-length Markov chains

In order to address this dimensionality problem, approaches have been proposed in PE where the model does not explicitly enumerate all possible states, but only implicitly represents them. The state space can be obtained by *executing* the model. This is akin, e.g., to Petri nets and BPMN models which generate, the traces. A well-known such approach, established within PE [7], Bioinformatics [38, 39], NLP [40], and data compression [41], is based on Variable-Length Markov Chains (VLMCs) [42, 43]. 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 Figure 1c.

Informally, VLMCs are able to implicitly represent Markov chains with only relevant memory such as in Figure 1c. Formally, VLMCs make use of a particular data structure used to track which part of the process history is relevant in each state to determine the outgoing probability distribution. This is encoded in the *context function*, hereby denoted by $\mathcal{C}$. Let us consider a trace of states $x_0x_1 \cdots x_n$, where x_0 is the initial state, and x_n the final one.[1] The context of this trace, i.e., $\mathcal{C}(x_0x_1 \cdots x_n)$, is the longest prefix $x_{n-k+1} \cdots x_n$ 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 states 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 so-called *probabilistic suffix tree* (PST) [44]. In what follows, we informally discuss how to use PSTs and VLMCs. We refer to [7] for details on the algorithms to synthesise them.

The PST of the program in Listing 1 is shown in Figure 2. Nodes are labeled with probability distributions of the next event, conditioned on the relevant history (i.e., the labels of the edges from the node to the root). The PST is traversed to compute the probability distribution of the next event. In particular, the traversal starts from the root, following the edge labeled with the latest produced event. If the target node has an outgoing edge labeled with the second-latest produced event, 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`. 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

[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.

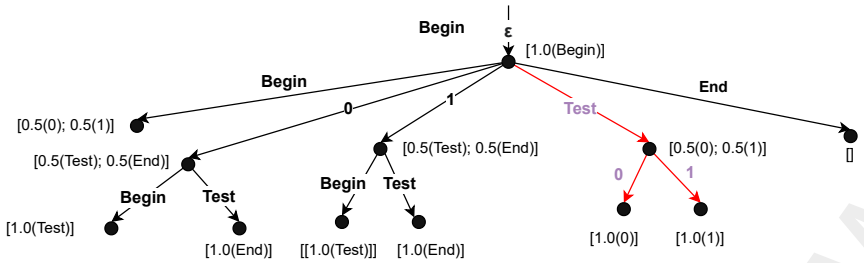


Fig. 2: PST modeling the behavior of program in Listing 1. Each edge is labeled with the name of an event emitted by the program while in each node the next symbol probability distribution is reported. With $\boldsymbol{\varepsilon}$ we denote the empty context (i.e., the first symbol of an events' sequence). Each path from the root to an edge identifies the next symbol distribution given the traversed edge symols.

`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`. We start by considering the latest event, `Test`, ignoring the previous ones. That is, we consider the red edge with label `Test` in Figure 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 Figure 1a. 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 event 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 length greater than two, this program can be described by using only memory on up to two steps before the newly generated one.

3.3 Stochastic conformance checking with VLMC

Here we will see how to use VLMCs and PSTs to enable VLMC-based stochastic conformance checking. We do this by introducing a notion of *likelihood* for VLMCs. In the theory of stochastic processes, the likelihood of a trace (actually a sequence) of events (or symbols, or states, depending on the domain) is a well-known notion [45]. It is connected to the probability that a trace is generated by a stochastic process. E.g., it is used in maximum likelihood estimation [46], where one searches for the model that best fits the data (e.g., the model with highest average likelihood for all traces).

We define the notion of likelihood for VLMCs as follows.

Definition 1 (Likelihood of a trace in a VLMC)**.** *Let $X = x_0 x_1 \dots x_n$ be a trace, and $\mathcal{M}$ be a VLMC with a corresponding PST and context function $\mathcal{C}$. The likelihood $\mathcal{L}(X)$ of observing the trace X in $\mathcal{M}$ is defined as:*

$$\mathcal{L}(X) = \prod_{i=1}^{n} Pr(x_{n+1} \mid \mathcal{C}(x_{n-k_i} \dots x_{n-1} x_n)) \tag{1}$$

where k_i is the length of the context used at step i.

In words, the likelihood of a trace is the product of the probabilities of having each event in the trace as next event 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 its PST in Figure 2. In particular, we compute $\mathcal{L}(\texttt{Begin}, \texttt{0}, \texttt{Test}, \texttt{0}, \texttt{End})$. We divide the trace in five prefixes, for which we compute the following conditional probabilities:

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

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

As a second example, we compute $\mathcal{L}(\texttt{Begin}, \texttt{0}, \texttt{Test}, \texttt{1}, \texttt{End})$. 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 prefixes, for which we compute the following conditional probabilities:

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

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

We can see that for the first trace, which belongs to the program and therefore "conforms" to it, we got a positive 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 and therefore "does not conform" to it we got 0 as likelihood (i.e., it has zero probability of being generated by the model). Armed with this, we are now in a position to define the notion of *VLMC-based stochastic conformance checking*.

Definition 2 (VLMC-based stochastic conformance checking)**.** *Let $\mathcal{M}$ be a VLMC, and X a trace. Then X is stochastic conformant to $\mathcal{M}$ if and only if $\mathcal{L}(X) > 0$ in $\mathcal{M}$.*

We remark that, for performing stochastic conformance checking, it suffices to consider the likelihood of a trace as a qualitative information: if it is greater than 0, then the trace is conformant, otherwise it is non-conformant. In contrast to EMD, this has the important benefit of not requiring the user to come up with any threshold in deciding for the conformance of a trace. We will see in the next sections that needing thresholds is an important limitation of EMD. Furthermore, comparing again with

EMD, we have that the notion of likelihood is directly defined for single traces and the model. We do not have to transform one or more traces as well as the model itself in a stochastic language. We will see that this is another limitation of EMD.

4 Numerical evaluation on program execution logs

In this section, we present a comprehensive numerical evaluation that showcases the capabilities of VLMC for stochastic conformance checking. We benchmark our approach against EMD, a method on which several state-of-the-art stochastic conformance checking approaches in PM are based (see, e.g., [3–5]). In this section , we consider a newly created dataset of program execution traces, in line with the example from Listing 1, which we make available to the community. Section 4.1 presents the scenario and the dataset. Sections 4.2–4.4 evaluate different aspects of our approach, showcasing merits and cons with respect to EMD-based ones.

All experiments were made using the implementation of EMD as available on pm4py and our own implementation of VLMC [2]. We used a common Apple M1 Pro equipped with 16GB of RAM.

4.1 Scenario: logs generated from two sorting algorithms

We consider logs generated running two implementations for the same algorithmic task. In particular, we consider as specific case study two famous sorting algorithms, namely Bubble sort and Selection sort [47]. We chose these two algorithms because they follow a similar logic, therefore the logs produced will be "similar", making it a complex and interesting test-bed for (stochastic) conformance checking. In particular, as we will see, the generated traces have same alphabet of events, possibly appearing with different frequencies and patterns. This setting is inspired by the one in [20], where the goal was to classify the right Microsoft Process Advisor[3] connector based on recorded user's behavior.

We generated a dataset of logs of *reference behavior* containing 10 000 executions of Bubble sort on lists with 6 randomly generated reals from 0 to 100. Then, we have built a dataset of logs for *validation behavior* of 20 000 new traces. In particular, 10 000 are *conformant*, meaning that they have been generated using again Bubble sort, and 10 000 *non-conformant*, i.e., obtained using Selection sort. The dataset is available at [17]. We remark that this dataset can be considered as one of the contributions of this paper.

Listing 2 reports the Python-like pseudocode of Bubble sort used as a reference process. Bubble sort is a simple sorting algorithm that repeatedly steps through a list, compares adjacent elements, and swaps them if they are in the wrong order. This process continues until the list is fully sorted. Specifically, the two loops in lines 5 and 7 implement the scan of the list's elements, while the test on line 9 is active when a swap is required.

[2] Code and replicability material is available at [17]

[3] https://learn.microsoft.com/en-gb/power-platform-release-plan/2021wave1/power-automate/process-advisor

Listing 2: Python-like pseducode for Bubble Sort with logging

```
1  prepareForNewTrace()
2  lst= #Randomly generated list with n entries
3  addEvent("Begin")
4  n = len(lst)
5  for i in range(n):
6      addEvent("Outer-loop")
7      for j in range(0,n-i-1):
8          addEvent("Inner-loop")
9          if lst[j] > lst[j+1]:
10             addEvent("Swap-required")
11             lst[j],lst[j+1] = lst[j+1],lst[j]
12 addEvent("End")
```

We use the same logging infrastructure as for Listing 1. The behavior of interest observed in the logs involves 5 activities:

- The begin and end of the algorithm (line 3 and 12, respectively),
- The execution of a new iteration of the outer loop (line 6),
- The execution of a new iteration of the inner loop (line 8),
- The establishment that a swap is required (line 10).

For example, the logs generated for the input lists `[1,3,5,4]` and `[8,2,4]` are shown in Table 2 (left).

Case id	*Timestamp*	*Activity*
B1	1	Begin
B1	2	Outer-loop
B1	3	Inner-loop
B1	4	Inner-loop
B1	5	Inner-loop
B1	6	Swap-required
B1	7	Outer-loop
B1	8	Inner-loop
B1	9	Inner-loop
B1	10	Outer-loop
B1	11	Inner-loop
B1	12	Outer-loop
B1	13	End
B2	1	Begin
B2	2	Outer-loop
B2	3	Inner-loop
B2	4	Swap-required
B2	5	Inner-loop
B2	6	Swap-required
B2	7	Outer-loop
B2	8	Inner-loop
B2	9	Outer-loop
B2	10	End

Case id	*Timestamp*	*Activity*
S1	1	Begin
S1	2	Outer-loop
S1	3	Inner-loop
S1	4	Inner-loop
S1	5	Inner-loop
S1	6	Outer-loop
S1	7	Inner-loop
S1	8	Inner-loop
S1	9	Outer-loop
S1	10	Inner-loop
S1	11	Swap-required
S1	12	Outer-loop
S1	13	End
S2	1	Begin
S2	2	Outer-loop
S2	3	Inner-loop
S2	4	Swap-required
S2	5	Inner-loop
S2	6	Outer-loop
S2	7	Inner-loop
S2	8	Swap-required
S2	9	Outer-loop
S2	10	End

Table 2: Logs generated for Bubble sort (left) and Selection sort (right) algorithms. Traces (B1 and S1) have been generated while sorting the input list `[1,3,5,4]` while traces B2 and S2 have been generated while sorting the input list `[8,2,4]`.

Listing 3: Python-like pseudocode for Selection Sort with logging

```
1  prepareForNewTrace()
2  lst= #Randomly generated list with n entries
3  addEvent("Begin")
4  n = len(lst)
5  for i in range(n):
6      min_index = i
7      addEvent("Outer-loop")
8      for j in range(i+1, n):
9          addEvent("Inner-loop")
10         if lst[j] < lst[min_index]:
11             min_index = j
12             addEvent("Swap-required")
13     lst[i],lst[min_index] = lst[min_index],lst[i]
14 addEvent("End")
```

Listing 3 shows the pseudocode of Selection sort used to generate the *not conformant* traces. Differently from Bubble sort, it divides the input list into two parts: a sorted sublist (initially empty) and an unsorted sublist (initially the whole input list). In each iteration (lines 5–13), it finds the smallest element in the unsorted sublist (lines 7–13) and swaps it with the first element of the unsorted sublist (line 13). This effectively expands the sorted sublist while shrinking the unsorted one. For example, the logs generated for the input lists `[1,3,5,4]`, and `[8,2,4]` are shown in Table 2 (right). Notably, in this specific case, the two algorithms generated the same number of events, albeit in a different order. However, this is not generally the case, as the two algorithms follow a different logic. By informally testing on a number of instances, we noted that differences in traces length increased when considering lists with 6 or more elements. This is the reason why in our experiments we consider lists of size 6.

4.2 Comparison of the two approaches trace by trace

We start the evaluation by comparing the ability of EMD and VLMC in identifying traces conformant to a given process (traces belonging to Bubble sort), and traces non conformant (traces belonging to Selection sort). In particular, in this section we do this *trace by trace*.

We check the conformance of each of the validation traces against all the reference traces. For each validation trace from Bubble sort, we check whether it is correctly classified as conformant (C_b), while for each validation trace from Selection sort we check whether it is correctly classified as not conformant ($\overline{C}_s$). We do this using both VLMC and EMD. The results are shown in Table 3. For VLMC, we get near-optimal performance: $C_b\%$=100 of conformant traces are identified as conformant; and $\overline{C}_s\%$=97.27 of non conformant traces are identified as non-conformant. Creating the VLMC once on the reference behavior took less than 10 seconds. Instead, as stated in the caption of the first two columns, running VLMC for computing the conformance of all 20 000 validation traces (RT) took overall just a few milliseconds.

VLMC, RT=1.30e-2s		*EMD, RT*=1.66e+4s, $EMD_b = 0.19$, $EMD_s = 0.26$							
No threshold required		*Th*=.001		*Th*=.19		*Th*=.26		*Th*=.33	
C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%
100.00	97.27	0.00	100.00	42.83	99.85	100.00	52.33	100.00	0.00

Table 3: Comparison of VLMC and EMD in detecting the conformance of 10 000 traces from *Bubble sort* (C_b), and the non-conformance of 10 000 traces from *Selection sort* ($\overline{C}_s$) for different thresholds (*Th.*). We consider traces one by one: for each trace, we compute its VLMC likelihood and EMD distance from a reference process (10 000 Bubble sort traces). A trace is conformant for EMD if the EMD distance is below the threshold, and for VLMC if the likelihood is greater than 0. Values EMD_b and EMD_s are the average EMD distance computed for each validation trace from Bubble sort and Selection sort, respectively. Finally, *RT* is the cumulative runtime for running VLMC and EMD, respectively, for each validation trace.

The results obtained for EMD, shown as well in Table 3, require a deeper discussion. Overall, they appear to be significantly worse in terms of accuracy and runtime than the ones obtained for VLMC. The caption specifies the average EMD value obtained for the 10 000 conformant traces (EMD_b) and the 10 000 non conformant ones (EMD_s). Such average is higher in the latter case, as expected. However, the two values are actually close, suggesting that EMD may not be particularly performant in establishing the conformance of single traces of this dataset. In fact, we use the EMD values computed for each trace to decide whether the trace is conformant or not. This operation requires a threshold against which comparing each EMD value. In Table 3 we consider 4 *reasonable* thresholds (*Th*): a very low value (0.001), the two mentioned average EMD values (EMD_b=.19 and EMD_s=.26), and a higher value (0.33) obtained by adding the difference among the two mean EMD values (0.07) to EMD_s. We can see that threshold 0.001 is too small. In fact, it leads to a trivial procedure that marks all traces as non-conformant. Likewise, 0.33 is too large. In fact, we get a trivial procedure that marks all traces as conformant.

The two intermediate thresholds show better accuracy, but still not comparable to the VLMC ones: 0.19 leads to erroneously marking about 57% of the conformant traces as non conformant, while 0.26 leads to erroneously marking about 48% of non conformant traces as conformant.

We have shown that EMD is very sensitive to the chosen threshold. Therefore, the need of carefully choosing such thresholds is a crucial aspect, and at the same time a limitation, of EMD-based approaches.

The caption of Table 3 provides the runtime for EMD. We can see that computing the EMD distance of each validation trace against all reference traces requires 6 orders of magnitude more time than in the VLMC case.

4.3 Comparison of the two approaches more traces at a time

We now move our attention to settings more in favour for EMD; namely, we do not check the conformance of single traces, but of groups of them. The results are shown

				Th=.001		*Th*=.19		*Th*=.26		*Th*=.33	
$\|Group\|$	EMD_b	EMD_s	RT	C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%
2	0.16	0.23	8.38e+3	0.00	100.00	97.20	99.20	100.00	8.14	100.00	0.00
5	0.12	0.21	3.41e+3	0.00	100.00	100.00	88.95	100.00	0.00	100.00	0.00
10	0.10	0.19	1.74e+3	0.00	100.00	100.00	51.70	100.00	0.00	100.00	0.00
20	0.08	0.18	9.16e+2	0.00	100.00	100.00	5.00	100.00	0.00	100.00	0.00
50	0.06	0.17	3.82e+2	0.00	100.00	100.00	0.00	100.00	0.00	100.00	0.00
200	0.03	0.16	1.29e+2	0.00	100.00	100.00	0.00	100.00	0.00	100.00	0.00
1000	0.01	0.15	3.04e+1	0.00	100.00	100.00	0.00	100.00	0.00	100.00	0.00
5000	0.01	0.15	8.29e+0	0.00	100.00	100.00	0.00	100.00	0.00	100.00	0.00
10000	0.01	0.15	5.65e+0	0.00	100.00	100.00	0.00	100.00	0.00	100.00	0.00

Table 4: Performance of EMD in detecting the conformance and non-conformance for the 20 000 reference traces and the 10 000 tested traces in Table 3. Differently from Table 3, we do not consider traces one by one, but we compute the EMD distance for groups of traces of varying size ($|Group|$). RT is the cumulative runtime for running EMD on each group of tested traces. EMD_b and EMD_s are as in Table 3.

in Table 4. We do not repeat analyses with VLMC, as it currently supports only single traces, and we already got very high accuracy for single traces.

To see how the group size impacts the performances of EMD, we consider groups of different size ($|Group|$). For each, we compute the average EMD obtained on the 10 000 Bubble sort validation traces (EMD_b), and on the 10 000 Selection sort ones (EMD_s). Column RT contains the runtime for computing all such EMD values. We can see that larger groups sizes lead to lower runtimes. This is because EMD is computed fewer times: from 5 000 times for group size 2, to only once for group size 10 000. In other words, EMD scales well with group size. However, even if we consider the best runtime obtained, we still get 3 orders of magnitude higher runtimes than in the VLMC case.

We note how EMD_b gets significantly smaller at the growing of the group size, and further away from EMD_s. For this reason, EMD has better accuracy when checking the conformance of more traces at a time. Notably, EMD_b almost halves from 0.19 for single traces (Table 3) to 0.10 when considering groups of 10 traces. EMD_b keeps decreasing further, but at a slower pace. Interestingly, we observe a curious phenomenon of EMD focusing on EMD_s, the average EMD distance from non-conformant traces. In fact, EMD values decrease also for non-conformant traces when increasing the group size. In particular, for groups of 10 traces it is about twice the corresponding EMD_b value, but it stabilizes at a much higher value ($\sim$0.15 for 200 traces or more).

Similarly to Table 3, we use the EMD values (of each group of traces) to mark traces as conformant or not by comparing them with chosen thresholds. We intentionally keep the four reference thresholds from Table 3 to study how the performances of EMD change with group size. For the two extreme thresholds we find similar patterns: 0.001 is too small, while 0.33 is too large. Differently from Table 3, now threshold 0.26 appears to be too large. Indeed, the average EMD values are smaller than in Table 3. The threshold 0.19 gives interesting results. Notably, using groups of size 2 or 5 leads to accuracy comparable, but slightly worse, than the VLMC case for single traces. Instead, further increasing the group size leads to worse performances due to the fact that EMD_b decreases.

These experiments show that EMD may have competitive performances when considering more traces at a time. However, this comes at higher computational costs and, more importantly, at the important extra burden of having to pick the *right* threshold value. For example, even without changing the dataset, the choice of the *right threshold* appears to be very sensitive to the group size. Finally, we remark that, in many cases, it might not be possible to compute in advance *dataset-specific* thresholds as we did. This is because, e.g., computing EMD_b and EMD_s requires knowing in advance all (validation) traces, as well as their conformance to reference traces. For example, this is not possible in online or streaming scenarios. In other words, picking the right threshold is crucial and difficult when using EMD for conformance checking.

4.4 Limits of VLMC: flexibility for incomplete logs

So far, we highlighted drawbacks of EMD concerning runtime and the sensitivity to the threshold. Here, we highlight drawbacks of VLMC concerning the representativeness and completeness of the event log. For doing this, we consider subsets of decreasing size of the original 10 000 reference traces. The 10 000 reference traces contain 720 different trace variants. Notably $6! = 720$, that is, the dataset is complete, because it contains all possible traces observable on this process. In fact, it can be shown that all initial lists with same ordering (e.g., `[1,2,3,4,5,6]` and `[11,12,13,14,15,16]`) will generate the same trace. Clearly, there are 6! possible such initial orderings.

The results are shown in Table 5. Columns $|Var.|$ and $Var.\%$ provide the absolute and percentage number of trace variants present in the selected traces. For example, the 8 000-case is still representative of the whole process, while the 100-case only contains about 12.92% of all trace variants.

For VLMC, we use group size 1 as in Table 3. Instead, for EMD we use group size 10, an intermediate one where EMD gave well-separated EMD_b and EMD_s values (see Table 4). For EMD we selected *reasonable* thresholds in line with what done in Table 3: we use a small value, 0.001, the EMD_b and EMD_s obtained for each subset (given in the last two columns), and a slightly larger value (0.28).

Table 5 tells us that using EMD_b as threshold gives a stable performance in correctly identifying conformant traces ($C_b\%$) within 52-58% across all the analyzed training set sizes. Instead, as highlighted in bold font, the performance of VLMC in correctly classifying conformant traces follows almost one-to-one the percentage of preserved trace variants. Therefore, the experiment suggests the following: if the log is representative for more than 50% of the traces of the process, one could opt for VLMC, while if the log contains less than 50% of the traces, and it is not important to check the conformance of single traces, one could go for EMD. This confirms one aspect of VLMC: it is particularly well suited for learning a language, but it may not generalize well to unseen traces. Both EMD with threshold EMD_b and VLMC are able to correctly classify non conformant traces with high accuracy (with a $\overline{C}_s\%$ equals to about 100 and 97/99, respectively).

For the other threshold values for EMD, we can see that 0.001 gives trivial results as in other tables because too restrictive, while the other values tend to be too permissive. Indeed, when using EMD_s (i.e., 0.19), about half of the non conformant traces are erroneously marked as conformant. Likewise, threshold 0.28 leads to trivial results.

Ref. traces			*VLMC* *No Th. required*		*EMD*, |*Group*|=10 *Th*=.001		*Th*=EMD_b		*Th*=EMD_s		*Th*=.28			
|*Traces*|	|*Var.*|	*Var.*%	C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%	C_b%	$\overline{C}_s$%	EMD_b	EMD_s
8000	720	**100.00**	**100.00**	97.27	0.00	100.00	56.20	100.00	100.00	49.20	100.00	0.00	0.10	0.19
4000	717	**99.58**	**99.61**	97.27	0.00	100.00	56.90	100.00	100.00	49.40	100.00	0.00	0.10	0.19
2000	680	**94.44**	**94.02**	97.71	0.00	100.00	56.20	100.00	100.00	49.20	100.00	0.00	0.10	0.19
1000	549	**76.25**	**74.96**	97.97	0.00	100.00	52.50	100.00	100.00	51.40	100.00	0.00	0.10	0.19
500	371	**51.53**	**50.56**	98.80	0.00	100.00	56.90	100.00	100.00	50.30	100.00	0.00	0.10	0.19
250	216	**30.00**	**29.64**	99.16	0.00	100.00	58.30	100.00	100.00	47.80	100.00	0.00	0.10	0.19
100	93	**12.92**	**12.73**	99.89	0.00	100.00	53.00	100.00	100.00	50.80	100.00	0.00	0.10	0.19

Table 5: Comparison of VLMC and EMD in detecting the conformance and non-conformance for the 20 000 tested traces in Table 3. Differently from Table 3, we do not consider all 10 000 reference traces and their 720 variants, but selections of different size (*Ref. traces*). For VLMC we considered the validation traces one by one, while for EMD we use group size 10 which is intermediate and shows well-separated EMD_b and EMD_s in Table 4.

5 Numerical evaluation on classic PM event log

In this section, we evaluate our approach on a real-life event log popular within PM. Namely, we consider the well-known Road Traffic Fine Management Process [16]. In Section 5.1 we briefly discuss the event log structure and an extension we do on it, while in Section 5.2 we report the numerical evaluation of our approach against EMD. In this case, we focus only on experiments *trace-by-trace* as in Section 4.2.

5.1 The dataset

The Road Traffic Fine Management dataset [16] 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 [48–56]. 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 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 conformance checking, we synthetize non-coformant traces by randomly altering part of the original ones. In particular, we considered 80% of the traces as the *reference behavior*, while a new set of logs was constructed for *validation behavior* using the following procedure. To test the ability to correctly detect conformant traces, we included the remaining 20% of the original dataset. To evaluate the ability to detect non-conformant behavior, we made a copy of such remaining 20% of traces, and we added *random noise* to them. In particular, we considered a

VLMC, RT=1.20e-2*s*		*EMD, RT*=1.25e+3*s*, EMD_o=0.44, EMD_m=0.63							
No Th. required		*Th*=.001		*Th*=.44		*Th*=.63		*Th*=.82	
C_o%	$\overline{C}_m$%	C_o%	$\overline{C}_m$%	C_o%	$\overline{C}_m$%	C_o%	$\overline{C}_m$%	C_o%	$\overline{C}_m$%
99.95	95.08	0.00	100.00	45.09	100.00	99.40	51.71	100.00	0.01

Table 6: Comparison of VLMC and EMD in detecting the conformance of 29 994 (20%) roadlines fines from the dataset (C_o), and the non-conformance of modified fines obtained from the considered 29 994 ones by randomly duplicating events. As in Table 3, we consider fines one by one: for each fine, we compute its VLMC likelihood and EMD distance from a reference process (the remaining 80% of fines in the dataset). For EMD we use thresholds selected as in Table 3, starting from the average EMD computed for the two groups of tested fines (EMD_o a EMD_m, respectively). RT is the cumulative runtime for running VLMC and EMD on each of the tested fines.

randomized version of *trace with additional event*, one of the *classic notions of noise in PM* described in Chapter 5.1.1 of [57]. 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 4, adding the new events in random positions. It is important to note that this method allowed us to generate non-conformant traces that are non-trivial to detect. The set of activities appearing in the conformant and non-conformant traces is identical, but they differ in their patterns.

5.2 Comparison of the two approaches trace by trace

In this section, we discuss the accuracy and runtime of conformance checking using VLMC and EMD. We do this *trace by trace*, similarly to Section 4.2. Specifically, we assess the conformance of each validation trace against all reference traces. For each validation trace from the *original* 20% ones, we verify whether it is correctly classified as conformant (C_o). Instead, for each *modified* validation trace we check whether it is correctly classified as non-conformant ($\overline{C}_m$).

The results are presented in Table 6. For VLMC, we achieve near-optimal accuracy: C_o%=99.95 of conformant traces are correctly identified as conformant. Similarly, $\overline{C}_m$%=95.08 of non-conformant traces are correctly identified as non-conformant. Additionally, as noted in the caption of the first two columns, running VLMC to compute the conformance of all 59 988 validation traces (RT) took only about 10 milliseconds in total. Instead, learning once the VLMC from the reference traces took less than 10 seconds.

The results obtained for EMD confirm our observations from Table 3. Overall, EMD's accuracy and runtime for single traces are significantly worse compared to VLMC. In the caption, we report the average EMD value obtained for the conformant traces (EMD_o) and the non-conformant ones (EMD_m). As expected, the average EMD is higher for the non-conformant traces, confirming EMD's ability to spot the deviations in the modified dataset. However, as for the sorting algorithms dataset, the two values are close, suggesting limited capabilities of EMD in distinguishing the two sets trace-by-trace.

EMD			
Th=.40		*Th*=.48	
$C_o\%$	$\overline{C}_m\%$	$C_o\%$	$\overline{C}_m\%$
43.79	100.00	52.02	98.53

Table 7: Same as Table 6, but focusing only on EMD for a different selection of thresholds. We consider computing the average EMD value for the conformant (original) fines, EMD_o, as a sort of training. We then use this information to select two threshold values close to it: $EMD_o \pm 10\%$.

As discussed in Section 4, in order to use EMD for conformance checking it is necessary to choose appropriate thresholds. In Table 6 we selected four threshold values following the same approach as in Table 3: a very low value (0.001), the two average EMD values mentioned earlier ($EMD_o = 0.44$ and $EMD_m = 0.63$), and a higher value (0.82) obtained by adding the difference between the two mean EMD values (0.19) to EMD_m. It is evident that a threshold of 0.001 is too small, resulting in a trivial procedure that marks all traces as non-conformant. Similarly, a threshold of 0.82 is too large, causing all traces to be marked as conformant. The two intermediate thresholds yield better performance, though still not comparable to VLMC. A threshold of 0.44 results in approximately 55% of conformant traces being mistakenly marked as non-conformant, while a threshold of 0.63 results in nearly 50% of non-conformant traces being erroneously classified as conformant.

These results highlight the limitation of EMD-based approaches and their high sensitivity to the choice of thresholds. Moreover, we observe that computing the EMD distance for each validation trace against all reference traces requires six orders of magnitude more time than for the VLMC approach.

To further investigate the impact of different thresholds on EMD accuracy, Table 7 studies it for two additional thresholds. Specifically, we used the computed average EMD value for the conformant fines (EMD_o) as a basis for selecting two threshold values close to it: $EMD_o \pm 10\%$. The results confirm once again the sensitivity of EMD-based conformance checking to the chosen thresholds. This further supports the use of VLMC in this context, as it provides high accuracy without requiring manual intervention to set threshold values.

6 Conclusion and future works

We introduced a novel approach to stochastic conformance checking inspired by techniques from software performance engineering (PE) based on the synthesis (i.e., mining) of Markovian processes from program execution traces (i.e., from logs). In particular, we use state-of-the-art techniques from PE based on higher-order Markovian models known as Variable-length Markov Chains (VLMC). We equip VLMCs with an easy-to-compute and accurate notion of likelihood used to establish whether a trace belongs to the mined model, or not. We used this notion for stochastic conformance checking. We also consider state-of-the-art approaches to stochastic conformance checking from prcoess mining (PM) based on the well-known Earth Mover's

Distance (EMD). We studied pros and cons of VLMC- and EMD-based stochastic conformance checking. In our experiments, we used both a newly created dataset from the PE field, and a classic one from PM. We found that VLMC often outperforms EMD-based ones in runtime and accuracy. This is particularly evident when checking the stochastic conformance of single traces (i.e., when evaluating the conformance trace by trace). In fact, our experiments suggest that EMD has limited accuracy in this setting. Another issue of EMD is its high sensitivity to the choice of thresholds to decide when a trace is non-conformant. Our experiments have demonstrated how even small changes in such thresholds may lead to drastically different results. In contrast, VLMC does not require any threshold.

On the other hand, our experiments suggested limitations in the flexibility of VLMC in handling incomplete logs. In particular, the ability of VLMC in correctly marking conformant traces appears to be closely correlated to the completeness of the logs (i.e., the percentage of trace variants contained in the logs).

Future works. We see this work as our first step towards bridging the gap between the PM and PE communities, fostering cross-fertilizations and opening new avenues for applications of stochastic conformance checking. For example, the improved accuracy in stochastic conformance of single traces may enable the stochastic extension of online conformance checking techniques (e.g., [18, 19]), and enable tasks of multi-labeled classification: *to which stochastic process belongs a given trace?* (e.g., [20]).

We also foresee research proposals focusing on making VLMC-like approaches "closer to PM". One important aspect is enhancing the flexibility of our approach to better handle incomplete logs or partially matching traces, settings of interest in PM. We have discussed how VLMCs give us precise likelihood (or probability) for a trace to be generated by the model. 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 encoding complex dependencies on the memory of the model. One can make VLMCs more flexible by pruning such computation to fewer sub-traces, i.e., by decreasing the amount of memory considered for a trace. This will lead to computing less precise but more permissive likelihood. A direction to pursue is evaluating how this may help to handle incomplete logs. 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, Declare or DCR Graphs: 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 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.

One dataset contained several repeated trace variants. In stochastic-centric approaches like ours, such "replicas" are important information used to fine tune models, making them able to estimate the probability of having a given trace. This paves the way for approaches comparing the *stochastic importance* of traces. For the other dataset, instead, we considered traces *with noise*. In particular, we considered one of

the "classic notions of noise in PM", namely *trace with additional event* (see Chapter 5.1.1 of [57]). Other notions of noise have been considered in the PM literature. We will extend our study to consider such notions to understand which ones affect most our approach. VLMCs are discrete-time, that is, they are higher-order discrete-time Markov chains. In future works, we might 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 one in [58], able to express complex constraints on data and resources.

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.

References

[1] Aalst, W.M.P.: Process Mining: Data Science in Action, 2nd edn. Springer, Germany (2016). https://doi.org/10.1007/978-3-662-49851-4

[2] Carmona, J., Dongen, B., Weidlich, M.: In: Aalst, W.M.P., Carmona, J. (eds.) Conformance Checking: Foundations, Milestones and Challenges, pp. 155–190. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08848-3_5

[3] Leemans, S.J.J., van der Aalst, W.M.P., Brockhoff, T., Polyvyanyy, A.: Stochastic process mining: Earth movers' stochastic conformance. Information Systems **102**, 101724 (2021) https://doi.org/10.1016/j.is.2021.101724

[4] Rocha, E.G., Leemans, S.J.J., Aalst, W.M.P.: Stochastic conformance checking based on expected subtrace frequency. In: 2024 6th International Conference on Process Mining (ICPM) (2024)

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

[6] Rüschendorf, L.: The wasserstein distance and approximation theorems. Probability Theory and Related Fields **70**(1), 117–129 (1985)

[7] Incerto, E., Napolitano, A., Tribastone, M.: 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, pp. 1–8. IEEE, USA (2020). https://doi.org/10.1109/MASCOTS50786.2020.9285947

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

[9] Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: A survey **30**(5), 295–310 (2004)

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

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

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

[13] Mazeroff, G., Gregor, J., Thomason, M., Ford, R.: Probabilistic suffix models for api sequence analysis of windows xp applications. Pattern Recognition **41**(1), 90–101 (2008)

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

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

[16] De Leoni, M., Mannhardt, F.: Road traffic fine management process. Eindhoven University of Technology, Dataset **284** (2015)

[17] Incerto, E., Vandin, A., Sarv Ahrabi, S.: Stochastic Conformance Checking based on Variable-length Markov Chains: Reaplication Package. Zenodo (2024). https://doi.org/10.5281/zenodo.13348234

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

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

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

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

[22] Leemans, S.J.J., Maggi, F.M., Montali, M.: Enjoy the silence: Analysis of stochastic petri nets with silent transitions. Information Systems **124**, 102383 (2024) https://doi.org/10.1016/j.is.2024.102383

[23] Augusto, A., Armas-Cervantes, A., Conforti, R., Dumas, M., Rosa, M.L.: Measuring fitness and precision of automatically discovered process models: A principled and scalable approach. IEEE Transactions on Knowledge and Data Engineering **34**(4), 1870–1888 (2022) https://doi.org/10.1109/TKDE.2020.3003258

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

[25] Burke, A., Leemans, S.J.J., Wynn, M.T.: Stochastic process discovery by weight estimation. In: Leemans, S., Leopold, H. (eds.) Process Mining Workshops, pp. 260–272. Springer, Cham (2021)

[26] Leemans, S.J.J., Fahland, D., Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) Business Process Management Workshops, pp. 66–78. Springer, Cham (2014)

[27] Chapela-Campa, D., Benchekroun, I., Baron, O., Dumas, M., Krass, D., Senderovich, A.: Can i trust my simulation model? measuring the quality of business process simulation models. In: Di Francescomarino, C., Burattin, A., Janiesch, C., Sadiq, S. (eds.) Business Process Management, pp. 20–37. Springer, Cham (2023)

[28] Brockhoff, T., Uysal, M.S., Aalst, W.M.P.: Time-aware concept drift detection using the earth mover's distance. In: 2020 2nd International Conference on Process Mining (ICPM), pp. 33–40 (2020). https://doi.org/10.1109/ICPM49681.2020.00016

[29] Burke, A., Leemans, S.J.J., Wynn, M.T.: Discovering stochastic process models by reduction and abstraction. In: Buchs, D., Carmona, J. (eds.) Application and Theory of Petri Nets and Concurrency, pp. 312–336. Springer, Cham (2021)

[30] Bolch, G., Greiner, S., De Meer, H., Trivedi, K.S.: Queueing Networks and

Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. John Wiley & Sons, Hoboken, NJ (2006)

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

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

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

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

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

[36] Ching, W.K., Fung, E.S., Ng, M.K.: Higher-order markov chain models for categorical data sequences. Naval Research Logistics (NRL) **51**(4), 557–574 (2004)

[37] Ever, E., Gemikonakli, O., Kocyigit, A., Gemikonakli, E.: A hybrid approach to minimize state space explosion problem for the solution of two stage tandem queues. Journal of Network and Computer Applications **36**(2), 908–926 (2013)

[38] Dalevi, D., Dubhashi, D., Hermansson, M.: A new order estimator for fixed and variable length markov models with applications to dna sequence similarity. Statistical applications in genetics and molecular biology **5**(1) (2006)

[39] Cunial, F., Alanko, J., Belazzougui, D.: A framework for space-efficient variable-order markov models. Bioinformatics **35**(22), 4607–4616 (2019)

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

[41] Nunez-Yanez, J.L., Chouliaras, V.A.: A configurable statistical lossless compression core based on variable order markov modeling and arithmetic coding. IEEE Transactions on Computers **54**(11), 1345–1359 (2005)

[42] Ron, D., Singer, Y., Tishby, N.: The power of amnesia: Learning probabilistic automata with variable memory length. Machine learning **25**(2-3), 117–149 (1996)

[43] Bühlmann, P., Wyner, A.J., *et al.*: Variable length markov chains. The Annals of Statistics **27**(2), 480–513 (1999)

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

[45] Rohde, C.A., *et al.*: Introductory Statistical Inference with the Likelihood Function. Springer, Berlin, Heidelberg (2014)

[46] Myung, I.J.: Tutorial on maximum likelihood estimation. Journal of mathematical Psychology **47**(1), 90–100 (2003)

[47] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd Edition. MIT Press, Cambridge, MA (2009). http://mitpress.mit.edu/books/introduction-algorithms

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

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

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

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

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

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

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

[55] Aalst, W.M., Maggi, F.M.: Conformance checking of processes based on monitoring real behavior. In: International Conference on Business Process Management, pp. 61–78 (2014). Springer

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

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

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