Event-Log Augmentation under User-Defined Process Constraints

Alessandro Padella[1], Letizia Cimbro[1], and Massimiliano de Leoni[1]

University of Padua, Italy
alessandro.padella@unipd.it
letizia.cimbro@studenti.unipd.it
deleoni@math.unipd.it

Abstract. Event-log augmentation is a common procedure to mitigate the scarcity of real-world data and, by extension, in the field of process mining. Existing augmentation techniques predominantly focus on reproducing global process patterns but offer limited control over domain-specific constraints. In practice, augmented event-logs must often satisfy explicit rules to remain meaningful for subsequent analysis. This paper introduces a constraint-aware framework for event-log augmentation, which integrates probabilistic models of process behavior with automaton-based representations of user-defined constraints. The framework ensures that augmented logs preserve both the variability of real data and the compliance with imposed rules. We evaluate the approach on four different use cases, each evaluated under four progressively stricter sets of constraints. Results based on entropy and Control-Flow-Log-Distance demonstrate that our method maintains compliance while enforcing diversity, with a computational efficiency that remains competitive.

Keywords: Process Mining · Event-log Augmentation · Finite-State-Automaton · Generative Methods · User-defined Constraints

1 Introduction

Process mining aims to understand and improve processes by analyzing their transactional data, which describes how individual executions unfold [1]. Such transactional data are typically captured in event logs, i.e., collections of traces that each represent one execution of the process. A trace is a sequence of events, where each event records the start or completion of a specific activity, together with its timestamp and other event-related perspectives, such as resources executing the activities.

Process mining is feasible only when an event log is available. As noted by Zimmermann et al. [13], data availability remains one of the major challenges faced by both practitioners and researchers in the field. The problem is particularly severe for approaches based on machine- or deep-learning models, which are inherently "data hungry". In process mining, this is often the case for simulation, predictive process monitoring, and prescriptive analytics [9, 6]. These

techniques rely on AI-based models, such as neural networks, that require substantial amounts of training data. Consequently, when the original event log is small, these techniques cannot be applied effectively. This motivates the need for event-log augmentation methods that enrich the original set of traces with newly generated ones.

Recently, a large body of research has been put forward to augment event logs (cf. Section 2), but the setting is not supported in settings in which process analysts would like to impose constraints when generating new traces. When possible, imposing constraints have several advantages, enabling the enforcement of domain rules and semantics: this prevents the introduction of unrealistic behavior in the generated traces and ensures generated traces be meaningful and avoid the introduction of unrealistic behavior. This ultimately means that compliance is ensured in the augmented event logs, with evident improvements in the usefulness of the generated traces for downstream tasks. For instance, when an event log is augmented without imposing constraints, and subsequently used in prescriptive process analytics, recommendations can be provided that violate the process' constraints.

This paper provides a technique for augmenting event logs under process' constraints that is based on finite-state automata (FSA) theory [8]. In a nutshell, process constraints are given as a set of finite-state automata. Given an event log $\mathcal{L}$ to be augmented, a stochastic finite-state automaton is generated to represent the behavior observed in $\mathcal{L}$. This stochastic finite-state automaton is intersected with the constraint automata, thus obtaining a new stochastic finite state automaton that is only capable to generate traces compliant with the constraints.

Experiments were conducted with four real-life event logs and four different set of process constraints for each of them, and compared with a baseline where the non-compliant traces are filtered out before generating a stochastic finite-state automaton that does not enforce the constraints. From a theoretical viewpoint, it is worth highlighting that the baseline does not always guarantee that the constraints are not violated (cf. discussion at the end of Section 5.3). From the practical viewpoint, our proposal guarantees a higher level of generalization while the similarity to the original event logs is largely retained at the same level. Generalization is here meant to indicate variability, which can be computed through the entropy-based metrics proposed by Back et al. [3], which quantitatively capture the information contained in the event logs. Similarity is measured through the metrics of Control-flow Log Distance [4], which measures the average minimal edits required to align traces between the logs.

The remainder of this paper is organized as follows. Section 2 reviews the related literature on event-log augmentation, motivating the choice of Stochastic FSA. Section 3 introduces the necessary preliminaries, including foundational definitions and core concepts relevant to event log augmentation. Section 4 details the proposed constraint-aware augmentation framework. Section 5 discusses the experimental goals and scenarios, the evaluation metrics, the use cases em-

ployed in the experimentation, and also the results in Section 5.3. Finally, Section 6 concludes the paper and outlines directions for future research.

2 Related Works

Traditional event-log augmentation techniques, commonly used in domains like computer vision and natural language processing, often assuming stochastic independence between samples [10]. However, such assumptions do not hold for events in logs, which are inherently arranged in sequences, where the temporal and contextual dependencies are crucial [1]. This is illustrated in our previous work [11], where we actually report on our experience of using SMOTE [5] for event-log augmentation: its use has shown no benefit, if not even reducing F-score, when augmenting the event log for predictive process monitoring.

The above motivates the importance of specific techniques for event-log augmentation, which has recently gained momentum in the realm of process mining, aiming to mitigate the scarcity of real-world data and facilitate the application of data-greedy techniques such as simulation and predictive process monitoring (see, e.g., the literature analysis in [11]). This body of research work however does not feature the possibility to impose adherence to a set of user-defined constraints. Notably, the proposal by Graziosi et al. [7], based on CVAE, is the only that considers constraints, which however cannot be provided by users.

In [11], we compare the accuracy and generalization capabilities of our approach with existing event-log augmentation techniques, including recent deep-learning–based frameworks. The results show that the generative method based on stochastic finite-state automata, referred to as the baseline in that work, achieves performance comparable to or better than alternative techniques, when generating sequences of activities (cf. Table 2 in [11]). At the same time, it is up to 99% faster than the fastest alternative not relying on stochastic FSAs (cf. Table 7 in [11]). In addition to its competitive performance and efficiency, the stochastic FSA–based framework is inherently white-box, making the integration of process constraints considerably simpler compared to black-box deep-learning approaches. These considerations motivate our choice of the stochastic FSA–based method as the foundation upon which to develop our constraint-aware extension. Section 3 provides a brief introduction to the basic generative technique based on Stochastic FSA, namely before extending it to feature user-defined constraints.

3 Preliminaries

The starting point for a process mining-based system is an *event log*. An event log is a multiset of *traces*. Since this paper only focuses on the activity labels when traces are generated, each trace is abstracted as a sequence of activities, each describing a particular execution of a *process instance*:

Definition 1 (Traces & Event Logs). *Let $\mathcal{A}$ be the set of possible activities, and let $\mathcal{B}(\mathcal{A}^*)$ be the set of all multisets of sequences in $\mathcal{A}$. A trace $\sigma \in \mathcal{A}^*$ is defined as a finite sequence of activities. An event log $\mathcal{L} \in \mathcal{B}(\mathcal{A}^*)$ is a multiset of sequences of activities in $\mathcal{A}$.*

To capture the different states in which a non-completed trace can be, we use the *prefixes*, i.e., a sequence of events that represent the process execution up to a certain point. Given a trace $\sigma = \langle a_1, \ldots, a_n \rangle$, the set of possible prefixes is defined as: $prefix(\sigma) = \{\langle\rangle, \langle a_1 \rangle, \langle a_1, a_2 \rangle, \ldots, \langle a_1, \ldots, a_n \rangle\}$. Analogously, we are also interested in annotating the set of prefixes of the traces in a given event log. The multiset containing all the prefixes of all the traces in an event log will be referred as $prefix(\mathcal{L})$, i.e. $prefix(\mathcal{L}) = \uplus_{\sigma \in \mathcal{L}} prefix(\sigma)$.[1]

The remainder of this section illustrates the formal foundation of the log-augmentation framework based on stochastic final state automaton. We start introducing a $last_k$ function, helping the formalization of the framework.

Definition 2 (Last-k Function). *Given a trace σ and a positive natural number k. The $last_k$ function is defined as:*

$$last_k(\langle\rangle) = \langle\rangle$$

$$last_k(\langle a_1, \ldots, a_n \rangle) = \begin{cases} \langle\rangle & \text{if } \sigma = \langle\rangle, \\ \langle a_{n-k}, \ldots, a_n \rangle & \text{if } k < n, \\ \langle a_1, \ldots, a_n \rangle & \text{if } k \geq n \end{cases}$$

Building upon the $last_k$ function, we can now formally define the concept of log K-Finite State Automaton:

Definition 3 (K-Finite State Automaton (k-FSA)). *Let $\mathcal{L}$ be an event log defined over a set $\mathcal{A}$ of activities. Let k be a natural number. The Stochastic k-Finite Automaton is defined as $(S, \mathcal{A}, \delta, \langle\rangle, F)$, which denotes a final state automaton, consisting of*

- *a state set $S = \bigcup_{\sigma \in prefix(\mathcal{L})} last_k(\sigma)$,*
- *a transition set $\mathcal{A}$,*
- *a transition function $\delta : S \times \mathcal{A} \nrightarrow S$ s.t.*
  - $\forall a \in \mathcal{A}\ ; \langle a \rangle \in prefix(\mathcal{L}) \Rightarrow \delta(\langle\rangle, a) = \langle a \rangle$
  - $\forall \langle a_1, \ldots, a_n, a_{n+1} \rangle \in prefix(\mathcal{L})$. $\delta(last_k(\langle a_1, \ldots, a_n \rangle), a_{n+1}) = (last_k(\langle a_1, \ldots, a_{n+1} \rangle)$
- *the empty trace $\langle\rangle$ is the initial state,*
- *a set $F = \bigcup_{\sigma \in \mathcal{L}} last_k(\sigma)$ of final states.*

The following example illustrate how a 2-FSA is created starting from an event log:

[1] Symbols $\uplus$ indicates the union of multisets.

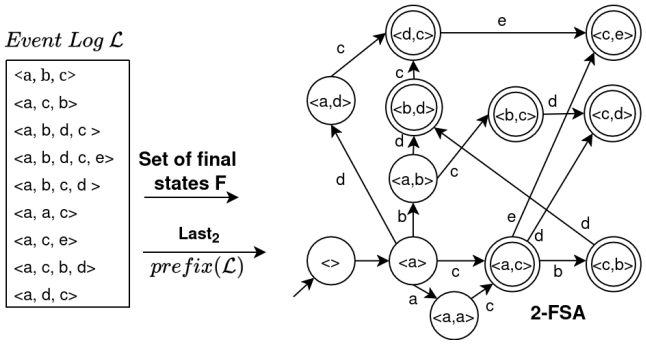

Fig. 1: Example of generation of a 2-FSA leveraging an event log $\mathcal{L}$ and a $last_2$ function.

Example 1 (2-FSA). An example of 2-FSA is depicted in Figure 1. Starting from an event log $\mathcal{L}$ composed of 9 complete traces, the multiset of prefixes $prefix(\mathcal{L})$ and the function $last_k$ are leveraged to infer the set of states $S = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle, \langle a, a\rangle, \langle a, c\rangle, \langle a, d\rangle, \langle b, c\rangle, \langle b, d\rangle, \langle c, b\rangle, \langle c, e\rangle, \langle d, c\rangle, \langle c, d\rangle, \langle c, e\rangle\}$. Later, the set of final states $F = \{\langle b, d\rangle, \langle d, c\rangle, \langle c, e\rangle, \langle b, c\rangle, \langle c, d\rangle, \langle a, c\rangle, \langle c, b\rangle\}$ is created from $\mathcal{L}$ and reported in the 2-FSA.

The transitions of a k-FSA can be annotated with the occurrence probability, thus yielding a Stochastic k-FSA:[2]

Definition 4 (Stochastic K-Finite State Automaton (Stochastic k-FSA)). *Let $\mathcal{L}$ be an event log defined over a set $\mathcal{A}$ of activities. Let k be a natural number. A Stochastic k-Finite Automaton is a tuple $(S, \mathcal{A}, \delta, \langle\rangle, F, P)$, where $(S, \mathcal{A}, \delta, \langle\rangle, F)$ is a k-FSA and $P : S \times \mathcal{A} \nrightarrow [0, 1]$ is a function that for each $(s, a) \in \mathrm{dom}(\delta)$, returns the probability $P(s, a)$ of firing transition (s, a).*

Note that for any final states $s \in F$, $0 \leq \sum_{a \in \mathcal{A}} P(s, a) \leq 1$. This occurs because the trace generation stops when reaching s with probability $1 - \sum_{a \in \mathcal{A}} P(s, a)$ and goes on with probability $\sum_{a \in \mathcal{A}} P(s, a)$.

Leveraging the defined probability function P, the Stochastic k-FSA can be traversed to stochastically generate sequences of symbols that conform to the behavior captured by the underlying finite-state automaton. Consequently, each generated sequence is guaranteed to be valid with respect to the automaton structure, while the stochastic nature of the transitions enables the realistic modeling and simulation of process variability observed. The probabilities are derived directly from the original event log, ensuring that the resulting model accurately reflects the empirical control-flow dynamics observed in the real data.

Given an event log $\mathcal{L}$ and its corresponding k-FSA $(S, \mathcal{A}, \delta, \langle\rangle, F)$, the probability function of the corresponding Stochastic k-FSA $(S, \mathcal{A}, \delta, \langle\rangle, F, P)$ is defined as follows. For each $(s, a) \in \mathrm{dom}(\delta)$, the probability $P(s, a)$ is the ratio between

[2] Given a function f, in the remainder, $\mathrm{dom}(f)$ denotes the domain of f.

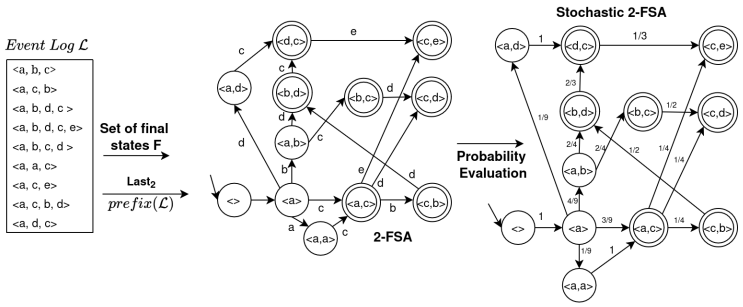

Fig. 2: Example of generation of a Stochastic 2-FSA, starting from the same event log proposed in Figure 1.

(i) the number of prefixes that ends with sequence s followed by a and *(ii)* the number of prefixes in which the trace suffix is s:[3]

$$P(s,a) \coloneqq \frac{\left|\biguplus_{\sigma' \in \text{prefix}(\mathcal{L}).\, \text{last}_{k+1}(\sigma') = s \oplus \langle a \rangle} \sigma'\right|}{\left|\biguplus_{\sigma' \in \text{prefix}(\mathcal{L}).\, \text{last}_{k}(\sigma') = s} \sigma'\right|} \tag{1}$$

The probabilistic transitions in the automaton are directly grounded in the observed frequency of activity successions in the event log. This data-driven probability assignment allows the automaton to generate new traces that not only comply with the structural rules encoded in the k-FSA but also preserve the stochastic properties of the original process behavior.

Let $\mathcal{Z} = (S, \mathcal{A}, \delta, q_0, F, P)$ be a Stochastic k-FSA. A trace $\sigma = \langle a_1, a_2, \cdots, a_n \rangle \in \mathcal{A}^*$ is admissible by $\mathcal{Z}$, and thus can be generated, iff there a state sequence $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots \xrightarrow{a_n} s_n$ such that $s_i = \delta(s_{i-1}, a_u) \in S$ and $s_n \in F$. If σ is admissible, the probability of generating σ is $\prod_{i=1}^{n} P(s_{i-1}, a_i)$.

Example 2 (Stochastic 2-FSA). In the example, also depicted in Figure 2, the event log $\mathcal{L}$ from Example 1 is used to construct a Stochastic 2-FSA. Using the multiset $prefix(\mathcal{L})$, transition probabilities are calculated according to the formula in Eq. 1. For instance, at the state $\langle a \rangle$, the probability of firing the transition labeled c is $\frac{3}{9}$. This is because the next states $\langle a, c \rangle$ appear in the 3 traces $\langle a, c, b \rangle$, $\langle a, c, e \rangle$, and $\langle a, c, b, d \rangle$, out of 9 occurrences of the state $\langle a \rangle$, since every trace starts with $\langle a \rangle$. In contrast, consider the state $\langle d, c \rangle$, which is followed only once by $\langle c, e \rangle$ in the log. Since $\langle d, c \rangle$ occurs 3 times in total, and 2 of these are terminal occurrences, the probability of transitioning with e is $\frac{1}{3}$, while the probability of terminating generation at this state is $\frac{2}{3}$.

[3] Symbol | is here overridden to be applied to multisets, namely to account for the sequence frequencies, and $\oplus$ is used to denote the concatenation of sequences.

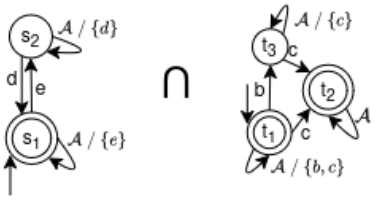

Fig. 3: Example of two constraint automata: $C_1 = \{$if d occurs, e has to occur later$\}$, and $C_2 = \{$if b occurs, c has to be in the trace$\}$

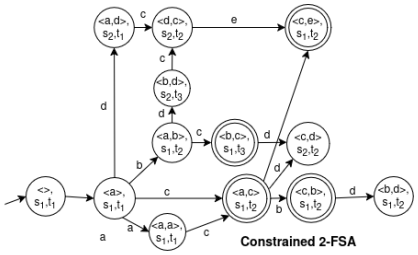

Fig. 4: Example of a Constrained 2-FSA.

4 The Framework Proposal

This section introduces the core framework for event-log augmentation under user-defined process constraints. These constraints are expected to be defined by process analysts in form of finite state automata:

Definition 5 (Constraint Automaton). *Let $\mathcal{A}$ be a finite set of activities. A Constraint Automaton over a set $\mathcal{A}$ of activities is a tuple $(Q, \mathcal{A}, \delta, q_0, F)$ where:*

- *Q is a finite set of states,*
- *$\delta : Q \times \mathcal{A} \nrightarrow Q$ is the transition function,*
- *$q_0 \in Q$ is the initial state,*
- *$F \subseteq Q$ is the set of final states.*

Note that the above definition of constraint automata is compliant with every constraint definition supported by the DECLARE language [2], although it is more general. Users can thus provide the constraints through the graphical notation of DECLARE. The following shows examples of constraint automata:

Example 3 (Constraint Automata). An example of two constraint automata is shown in Figure 3. It illustrates the two constraints: $C_1 = \{$if d occurs, e has to occur later$\}$, and $C_2 = \{$if b occurs, c has to be in the trace$\}$. The automaton for C_1 consists of two states, s_1 and s_2, where only s_1 is accepting. In contrast, the automaton for C_2 has three states: t_1 represents when neither b nor c has occurred; t_2 represents the situation where c has occurred (removing restrictions on b); and t_3 denotes that b has occurred but c has not yet appeared. The two automata will be then intersected to form a Constraint Automaton whose states are pairs drawn from the original automata's states. A state in this automaton is accepting if and only if both constituent states are accepting.

Definition 6 (Constrained k-FSA). *Let $\mathcal{A}$ be a set of activities, and let $\mathcal{C} = \{(Q_1, \mathcal{A}, \delta_1, q_{0,1}, F_1), \ldots, (Q_i, \mathcal{A}, \delta_i, q_{0,n}, F_n)\}$ be a set of constraint automata. The Constrained k-FSA is obtained by intersecting the Constraint Automata in $\mathcal{C}$ and a given k-FSA $(S, \mathcal{A}, \overline{\delta}, \langle\rangle, \overline{F})$ and consists in the automaton $(Q, \mathcal{A}, \delta, q_0, F)$ defined as follows:*

- $Q = S \times Q_1 \times Q_2 \times \cdots \times Q_n$ *is the product state space,*
- $q_0 = (\langle\rangle, q_{0,1}, q_{0,2}, \ldots, q_{0,n})$ *is the initial state,*
- *the transition function* $\delta : Q \times \mathcal{A} \not\to Q$ *is defined as follows: For each state* $(s, q_1, \ldots, q_n) \in Q$ *and activity* $a \in \mathcal{A}$*, if* $(s, a) \in \mathrm{dom}(\overline{\delta})$ *and* $(q_1, a) \in \mathrm{dom}(\delta_1)$ *and* $\ldots$ *and* $(q_n, a) \in \mathrm{dom}(\delta_n)$

$$\delta\big((s, q_1, \ldots, q_n),\, a\big) = \big(\overline{\delta}(s, a), \delta_1(q_1, a),\ \delta_2(q_2, a),\ \ldots,\ \delta_n(q_n, a)\big),$$

- *the set of final states is* $F = \overline{F} \times F_1 \times F_2 \times \cdots \times F_n$.

Example 4 (Constrained 2-FSA). The Constraint Automaton log, obtained intersecting the automaton C_1 and C_2 from the example in 3, is then intersected with the 2-FSA from the example in 1. The resulting automaton, referred to as the Constraint 2-FSA and depicted in Figure 4, has states defined as tuples composed of the states of the input automata. A tuple is a final state if and only if every component state from the input automata is final. In this case, the four identified final states are $(\langle c, e\rangle, s_1, t_2)$, $(\langle b, c\rangle, s_1, t_3)$, $(\langle a, c\rangle, s_1, t_2)$, and $(\langle c, b\rangle, s_1, t_2)$. Conversely, the states $(\langle c, d\rangle, s_2, t_2)$ and $(\langle b, d\rangle, s_1, t_2)$ have no possible outgoing transitions and are not final; such states are referred to as *dead ends*.

The use of a constrained k-FSA to augment an event log requires one to associate transitions with probabilities, yielding a Stochastic Constrained k-FSA:

Definition 7 (Stochastic Constrained k-FSA). *Let* $\mathcal{A}$ *be a set of activities, and let* $\mathcal{C}$ *be a set of constraint automata. Let* $\mathcal{K}$ *be a* k*-FSA, and let* $\mathcal{S} = (\mathcal{K}, P)$ *be an accordant Stochastic* k*-FSA. The Stochastic Constrained* k*-FSA of* $\mathcal{C}$ *and* $\mathcal{S}$ *is a tuple* $(Q, \mathcal{A}, \delta, q_0, F, P)$ *where* $(Q, \mathcal{A}, \delta, q_0, F)$ *is the Constrained* k*-FSA of* $\mathcal{C}$ *and* $\mathcal{K}$ *and* $P : Q \times \mathcal{A} \not\to [0, 1]$ *is a function that assigns a probability* $P(q, a)$ *to each pair* (q, a) *in the domain of* δ.

The function P_c of a Stochastic Constrained Automata $(Q_c, \mathcal{A}, \delta_c, q_{0_c}, F_c, P_c)$ of a set $\mathcal{C}$ of constraint automata and a stochastic k-FSA $(S, \mathcal{A}, \overline{\delta}, \langle\rangle, F_c, P)$ is computed as follows:

1. Assignment of a zero probability to transitions reaching a dead state. As also illustrated in Example 4, a Constrained k-FSA can contain some *dead* states q from which no final state is reachable, namely there is no sequence $\langle b_1, \ldots, b_n\rangle \in \mathcal{A}^*$ such that $\delta(\ldots\delta(\delta(q, b_1), b_2)\ldots, b_n) \in F_c$. First, we identify the set $Q_d \subset (Q_c \setminus F_c)$ of *dead* states. For each transition $(q, a) \in \mathrm{dom}(\delta_c)$ such that $\delta_c(q, a) \in Q_d$, $P_c(q, a) \coloneqq 0$.

2. Assignment the probabilities to non-dead state. At step 1, the probabilities are set to zero for all transitions that lead to a dead state. To keep consistencies, the probabilities of the other transitions need to be scaled up: recalling that Q_d identifies the dead states, for each transition $((s, \overline{q}), a) \in \mathrm{dom}(\delta_c)$ such that $\delta_c((s, \overline{q}), a) \notin Q_c$, its probability is set as follows:

$$P_c((s, \overline{q}), a) \coloneqq P(s, a) \cdot \frac{\sum_{a' \in \mathcal{A}.(s,a') \in \mathrm{dom}(\overline{\delta})} P(s, a)}{\sum_{a' \in \mathcal{A}.(s,a') \in \mathrm{dom}(\overline{\delta}) \wedge \delta_c((s,q),a') \notin Q_c} P(s, a)} \tag{2}$$

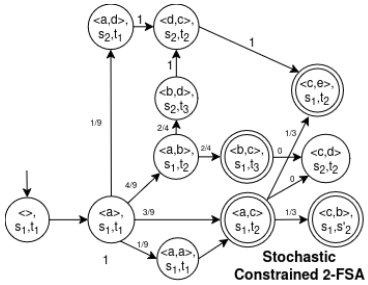

Fig. 5: Visual representation of the Stochastic-Constrained 2-FSA, obtained from the Constrained-2FSA in Figure 4 and the Stochastic 2-FSA in Figure 2.

Note how the ratio present in Equation 2 is always larger than or equal to one. If the probability of any transition $((s, q), a')$ is set to zero, the ratio is larger than one, thus distributing those probabilities to every transition that enables reaching a final state, proportionally to the original probabilities.

Example 5 (Stochastic Constrained 2-FSA). In this example, also depicted in Figure 5, the Stochastic 2-FSA created in example 2 and the Constrained 2-FSA created in example 4 are used to create the Stochastic Constrained 2-FSA. As reported in Definition 7, the construction of it implies two steps: in the first step the transitions reaching a dead state are assigned probability 0. This applied to the dead states $(\langle c, d\rangle, s_2, t_2)$ and $(\langle b, d\rangle, s_2, t_2)$ that have been assigned probability 0. Then, probabilities are assigned and rescaled.

5 Evaluation and Results

Section 5.1 details the use cases, while Section 5.2 illustrates the experimental methodology and the assessment metrics. Section 5.3 reports on the results.

5.1 Use Cases

The validity of the framework was assessed using four different processes from which an event-log was extracted, and a four different constraint has been set.

Purchasing. A process provided as part of the Fluxicon Disco tool and it is related to a purchase-to-pay[4].

BPI12. The log from a loan application process, used by the BPI challenge in 2012[5].

[4] https://fluxicon.com/academic/material

[5] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

Use Case	exp1	exp2	exp3	exp4
Purchasing	The process must start with *Create Purchase Requisition*.	A *Create Request for Quotation* must eventually follow a *Create Purchase Requisition*.	After *Create Quotation comparison Map*, *Analyze Quotation Comparison Map* must eventually occur.	The process must start with *Create Purchase Requisition*; a *Create Request for Quotation* must follow it; after *Create Quotation comparison Map*, *Analyze Quotation Comparison Map* must occur.
BPI12	The process must start with *WAfhandelen leads*.	After *WCompleteren aanvraag*, *WNabellen incomplete dossiers* must eventually occur. Also, *WNabellen incomplete dossiers* must exist in the process.	After *WNabellen incomplete dossiers*, *WBeoordelen aanvraag* must eventually occur.	The process must start with *WAfhandelen leads*; after *WCompleteren aanvraag*, *WNabellen offertes* must follow; then after *WNabellen offertes*, *WValideren aanvraag* must follow; *WNabellen incomplete dossiers* must exist.
BPI17W	The process must start with *WComplete application*.	The event *WCall incomplete files* must exist at least once.	The event *WValidate application* must exist at least once.	The process must start with *WComplete application* and *WValidate application* must occur exactly once.
Hospital	The event *DIMISSIONE* must precede *USCITA*.	Constraints for this case are defined in the extended version.	Constraints for this case are defined in the extended version.	Constraints for this case are defined in the extended version.

Table 1: The sets of constraints employed, denoted by *exp1* to *exp4*, during the experiments for the different use cases.

BPI17W. The log used by the BPI challenge in 2017 in which only the workflow-relevant activities have been maintained. It is provided by the same financial institution that provides the log employed in *BPI12*.[6]

Hospital. This process has been provided by an Italian hospital emergency department, and it is private.

Table 1 summarizes the sets of constraints applied to each use case in the experiments. These constraints dictate the permissible activity sequences in the augmented event logs, ensuring both domain compliance and meaningful process variability. The constraints related to exp2, exp3 and exp4 for the use case Hospital have been reported in the extended version, along with the implementation of the framework.[7]

5.2 Experimental Goals, Metrics and Comparison

Our Stochastic Constrained k-FSA only allows generating traces that do not violate the constraints, because that is true by construction (i.e., the intersection of automata). Therefore, it is unnecessary to verify this through evaluation.

The experimental evaluation aimed to compare our technique based on Stochastic Constrained k-FSA with a **baseline** in which the traces that violate the constraints are filtered out before generating a Stochastic k-FSA, which namely that does not enforce the constraints. The hypothesis is that our technique is better in generalizing than the **baseline**, because our technique still learns from the portions of the violating traces that comply the constraints, while the **baseline** would just ignore those traces altogether.

[6] https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
[7] https://github.com/Pado123/Constrained_data_aug

Hereafter, the generalization of an event log is measured via assessing its variability using an entropy-based metric derived from Shannon's information theory and adapted to the process mining. In particular, Back et al. introduces a relevant metric, also adopted in [11, 12]:

Definition 8 (Trace Entropy). *Let $\mathcal{L}$ be an event log, and let $\mathcal{L}(\sigma)$ be the cardinality of σ in $\mathcal{L}$, namely the number of traces equal to σ in $\mathcal{L}$. Recalling $|\mathcal{L}|$ be the number of traces in $\mathcal{L}$, the* trace entropy *of $\mathcal{L}$ is defined as:*

$$H_t(\mathcal{L}) = -\sum_{\sigma \in \mathcal{L}} \frac{\mathcal{L}(\sigma)}{|\mathcal{L}|} \cdot \log \frac{\mathcal{L}(\sigma)}{|\mathcal{L}|}$$

Intuitively, higher values of trace entropy indicates a higher variability, and thus generalization.

Orthogonally, the evaluation aimed to asses that the generated traces remain consistent with the original event log, to this extend we leverage the **Control-flow Log Distance** (CFLD) metric [4]. Given two event logs, $\mathcal{L}_1$ and $\mathcal{L}_2$, CFLD measures the average minimal distance required to transform each trace in $\mathcal{L}_1$ into a trace in $\mathcal{L}_2$, thereby quantifying behavioral similarity between logs. Our goal is to assess how similar the generated logs are in relation to specific scenarios. For a given set of constraints C, we derive a filtered event log $\mathcal{L}^{\mathcal{C}}_{test}$. We then compute the CFLD between the generated log, constrained by $\mathcal{C}$, and $\mathcal{L}^{\mathcal{C}}_{test}$, i.e. the removing from the event log that we individuate as test log the traces that are not compliant with the set of constraints C. This comparison elucidates the extent to which our generated data complies with the original event log under the imposed constraints.

In the reminder, our technique has been evaluated with $k = 3$ and $k = \infty$, which has been compared with the baseline with trace filtering $k = 3$ and $k = \infty$. The choice of $k = 3$ is linked to the finding in [11] where, as far as the experiments report, $k = 3$ has shown to provide a good balancing between similarity of the traces with those of the initial event logs, on the one hand, and capability to generalize beyond the sole traces that were observed, on the other hand.
The set of constraints were selected in an incremental manner: for each use case, the first set of constraints were the least restrictive, when the fourth were the most.

Training logs were built on the earliest 80% of the traces of the event logs of each use case. For each combination of training log $\mathcal{L}_{train}$ and set $\mathcal{C}$ of constraints, we trained the stochastic constrained 3-FSA and ∞-FSA (i.e., with $k = 3$ and $k = \infty$) on $\mathcal{L}^{\mathcal{C}}_{train}$, imposing the constraints in $\mathcal{C}$. For the baseline, as discussed, we first filtered out the traces of the training logs $\mathcal{L}^{\mathcal{C}}_{filter}$ that violated the constraints in $\mathcal{C}$; then we built a non-constrained stochastic 3-FSA and ∞-FSA (i.e., as defined in Definition 4) on $\mathcal{L}^{\mathcal{C}}_{filter}$.

For each combination of log and set of constraints, the remaining 20% of the traces constitute a test log. We also used the constrained and non-constrained 3-FSA and ∞-FSA to generate 10 synthetic event logs that contained the same number of traces as the respective test log. The generation of multiple synthetic

event logs for each combination of use case and set of constraints aimed to mitigate the effect of the generation stochasticity.

5.3 Experimental Results: Analysis and Final Discussion

Table 2 presents values of trace entropy measurements for our framework based on stochastic constrained k-FSA and for the baselines (i.e., filtering out traces and applying the non constrained k-FSA on the remaining traces), in all use cases and for all tested sets of constraints. The first two columns identify the use case and the experiment; the next four columns report results for various scenarios in terms of trace entropy. Finally, the last column reports the values of trace entropy of the original test sets for the event logs of the different use cases. Additionally, for each combination of method and set of constraints, the table reports in brackets the percentage of how the entropy is reduced with respect to that of the original test log. The last row of Table 2 illustrates the average reduction of entropy. The reduction of trace entropy in all scenarios, namely our Stochastic Constrained FSA and the baseline, is to be expected because the application of the constraints naturally restricts the set of traces that can be generated. So is it natural that the entropy's reduction is higher for $k = \infty$ because basically the ∞-FSA does not enable generating traces beyond those already observed. However, both for $k = 3$ and for $k = \infty$, this reduction is significant lower for our stochastic constrained k-FSA: this is because the stochastic constrained k-FSA still uses the portions of violating traces that do not violate the constraints, instead of filtering out the traces altogether.

Of course, generalization is just one of the two forces to consider when generating traces: similarity to the original traces is the second. In our experimental setting, similarity was measured using the CFLD metric, which was applied on the test event logs from which we removed the traces violating the respective constraints. Results are reported in Table 3 where the last row illustrates the average CFLD values per column. The averages show negligible differences among the CFLD values per method: they are all between 0.14 and 0.17. Recall that lower values are to be preferred. It is also expected that CFLD values are lower for $k = \infty$, which indeed can only generate traces that are observed in the test event logs.

In summary, the comparison of differences of values for the trace-entropy and CFLD metrics illustrate that our approach based on Stochastic Constrained k-FSA guarantees a significant higher level of generalization, if compared with the baselines, at a very negligible cost of reducing similarity.

Finally, Table 4 presents the average computational time of our proposal and the baseline for $k = 3$ or $k = \infty$. For $k = 3$ and $k = \infty$, the additional computation time is 4.5% and and 4.0%. This is certainly feasible, both as absolute time and its percentage, and is largely motivated by the need to compute the intersection of finite-state automata.

We conclude highlighting that if k is not set to infinity, the approach of filtering violating traces followed by generating traces from an unconstrained stochastic k-FSA does not guarantee that the resulting traces comply with the

		$k = 3$		$k = \infty$		
Use Case	**Exp**	**Stochastic Constrained 3-FSA**	**baseline**	**Stochastic Constrained ∞-FSA**	**baseline**	**Test H_t**
Purchasing	exp1	1.19 (0.84%)	1.17 (2.50%)	1.18 (1.67%)	1.18 (1.67%)	1.20
	exp2	1.15 (4.17%)	1.15 (4.17%)	1.16 (3.33%)	1.12 (6.67%)	
	exp3	1.18 (1.67%)	1.18 (1.67%)	1.18 (1.67%)	1.07 (10.83%)	
	exp4	1.15 (4.17%)	1.15 (4.17%)	1.15 (4.17%)	1.06 (11.67%)	
BPI12W	exp1	0.84 (4.55%)	0.72 (18.18%)	0.72 (18.18%)	0.72 (18.18%)	0.88
	exp2	0.54 (38.64%)	0.50 (43.18%)	0.50 (43.18%)	0.50 (43.18%)	
	exp3	0.56 (36.36%)	0.56 (36.36%)	0.56 (36.36%)	0.56 (36.36%)	
	exp4	0.52 (40.91%)	0.47 (46.59%)	0.52 (40.91%)	0.47 (46.59%)	
BPI17W	exp1	0.61 (6.15%)	0.61 (6.15%)	0.61 (6.15%)	0.61 (6.15%)	0.65
	exp2	0.64 (1.54%)	0.63 (3.08%)	0.64 (1.54%)	0.60 (7.69%)	
	exp3	0.65 (0.00%)	0.64 (1.54%)	0.64 (1.54%)	0.49 (24.62%)	
	exp4	0.58 (10.77%)	0.52 (20.00%)	0.53 (18.46%)	0.42 (35.38%)	
Hospital	exp1	1.27 (0.00%)	1.26 (0.79%)	1.25 (1.57%)	1.26 (0.79%)	1.27
	exp2	1.23 (3.15%)	1.19 (6.30%)	1.18 (7.09%)	1.10 (13.39%)	
	exp3	1.23 (3.15%)	1.14 (10.24%)	1.12 (11.02%)	1.09 (14.17%)	
	exp4	1.22 (3.94%)	1.14 (10.24%)	1.12 (11.02%)	0.87 (31.50%)	
Average		8.22%	13.44%	12.99%	19.30%	

Table 2: Trace entropy values for different use cases and constraint sets. The last column shows the trace entropy of the original test set for comparison. Values indicate trace entropies, with percentages in parentheses representing the entropy loss relative to the test set entropy.

specified constraints. The reason is that the Stochastic k-FSA only consider the last k activities, whereas the verification of a constraint may require reasoning over the entire sequence. As an example, consider an event log $\mathcal{L}_{ex} = \{\langle b, c, d, e, a\rangle, \langle a, b, c, d, e\rangle\}$, and a constraint that imposes that a must eventually occur in the trace. Let k be three. Both of traces do not violate the constraint, and thus they are retained to build the Stochastic 3-FSA. However, the obtained Stochastic 3-FSA allows the generation of trace $\langle b, c, d, e\rangle$, which violates the constraint.

6 Conclusions

This paper presents a novel constraint-aware framework for event-log augmentation, integrating user-defined process constraints, expressed as finite-state automata, directly into the generative process based on them. The approach ensures strict compliance of all generated traces with the specified constraints while preserving trace variability, which is essential for effective process mining tasks such as simulation and predictive monitoring. Empirical evaluation across four use cases demonstrates a superior compliance-variability trade-off compared to baselines that filter non-compliant traces, as validated by entropy-based metrics

		$k = 3$		$k = \infty$	
Use Case	**Exp**	**Stochastic Constrained 3-FSA**	**baseline**	**Stochastic Constrained ∞-FSA**	**baseline**
Purchasing	exp1	0.26	0.25	0.27	0.27
	exp2	0.25	0.24	0.29	0.26
	exp3	0.27	0.25	0.27	0.24
	exp4	0.27	0.25	0.31	0.27
BPI12W	exp1	0.13	0.12	0.07	0.07
	exp2	0.15	0.15	0.14	0.14
	exp3	0.13	0.10	0.09	0.08
	exp4	0.19	0.12	0.19	0.14
BPI17W	exp1	0.13	0.12	0.12	0.11
	exp2	0.14	0.14	0.14	0.14
	exp3	0.13	0.13	0.14	0.13
	exp4	0.21	0.20	0.20	0.19
Hospital	exp1	0.10	0.09	0.02	0.02
	exp2	0.13	0.13	0.03	0.03
	exp3	0.13	0.16	0.05	0.05
	exp4	0.15	0.16	0.08	0.08
Average		0.17	0.16	0.15	0.14

Table 3: CFLD values per use case and experiment compared with the original test log, once the trace violating constraints have been removed.

Stochastic Constrained 3-FSA	**baseline k=3**	**Stochastic Constrained ∞-FSA**	**baseline $k = \infty$**
182	174	156	150

Table 4: Average computational time over 10 generations for four use cases and four sets of constraints. Results are reported in terms of seconds.

and Control-flow Log Distance. The framework's foundation in FSA constructions yields computational efficiency and broad applicability, facilitating extensions to additional event-log perspectives including resources and timestamps. Future work will investigate the learning of soft constraints and applications in predictive/prescriptive analytics, additional investigations will address the generation of multiple perspectives on resource execution, aligned with the starting and ending timestamps of corresponding activities.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Berlin: Springer-Verlag (2011)
2. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science – Research and Development **23**(2), 99–113 (2009). https://doi.org/10.1007/s00450-009-0057-9

3. Back, C.O., Debois, S., Slaats, T.: Entropy as a measure of log variability. J. Data Semant. **8**(2), 129–156 (2019). https://doi.org/10.1007/S13740-019-00105-3
4. 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: Business Process Management. pp. 20–37. Springer Nature Switzerland, Cham (2023)
5. Chawla, N., Bowyer, K., Hall, L., Kegelmeyer, W.: Smote: Synthetic minority over-sampling technique. J. Artif. Intell. Res. (JAIR) **16**, 321–357 (06 2002). https://doi.org/10.1613/jair.953
6. Di Francescomarino, C., Ghidini, C.: Predictive Process Monitoring, pp. 320–346. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-08848-3_10
7. Graziosi, R., Ronzani, M., Buliga, A., Di Francescomarino, C., Folino, F., Ghidini, C., Meneghello, F., Pontieri, L.: Generating multiperspective process traces using conditional variational autoencoders. Process Science **2**, 8 (2025). https://doi.org/10.1007/s44311-025-00017-5
8. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
9. Mehdiyev, N., Fettke, P.: Explainable Artificial Intelligence for Process Mining: A General Overview and Application of a Novel Local Explanation Approach for Predictive Process Monitoring, pp. 1–28. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-64949-4_1
10. Mumuni, A., Mumuni, F.: Data augmentation: A comprehensive survey of modern approaches. Array **16**, 100258 (2022). https://doi.org/https://doi.org/10.1016/j.array.2022.100258
11. Padella, A., Vinci, F., de Leoni, M.: An experimental comparison of alternative techniques for event-log augmentation (2025), https://arxiv.org/abs/2511.01896
12. van Straten, S., Padella, A., Hassani, M.: Leveraging data augmentation and siamese learning for predictive process monitoring. arXiv preprint (2025). https://doi.org/10.48550/arXiv.2507.18293, arXiv:2507.18293
13. Zimmermann, L., Zerbato, F., Weber, B.: What makes life for process mining analysts difficult? a reflection of challenges. Software and Systems Modeling pp. 1–29 (11 2023). https://doi.org/10.1007/s10270-023-01134-0