Stochastic Directly-Follows Process Discovery Using Grammatical Inference

Hanan Alkhammash[1] (✉), Artem Polyvyanyy[1], and Alistair Moffat[1]

The University of Melbourne, Victoria 3010, Australia
halkhammash@student.unimelb.edu.au
{artem.polyvyanyy;ammoffat}@unimelb.edu.au

Abstract. Starting with a collection of traces generated by process executions, process discovery is the task of constructing a simple model that describes the process, where simplicity is often measured in terms of model size. The challenge of process discovery is that the process of interest is unknown, and that while the input traces constitute positive examples of process executions, no negative examples are available. Many commercial tools discover Directly-Follows Graphs, in which nodes represent the observable actions of the process, and directed arcs indicate execution order possibilities over the actions. We propose a new approach for discovering sound Directly-Follows Graphs that is grounded in grammatical inference over the input traces. To promote the discovery of small graphs that also describe the process accurately we design and evaluate a genetic algorithm that supports the convergence of the inference parameters to the areas that lead to the discovery of interesting models. Experiments over real-world datasets confirm that our new approach can construct smaller models that represent the input traces and their frequencies more accurately than the state-of-the-art technique. Reasoning over the frequencies of encoded traces also becomes possible, due to the stochastic semantics of the action graphs we propose, which, for the first time, are interpreted as models that describe the stochastic languages of action traces.

Keywords: Process mining, stochastic process discovery, directly-follows graphs

1 Introduction

Process mining is a discipline that studies data-driven methods and techniques to analyze and optimize processes by leveraging the event data extracted from information systems during process execution. Process mining approaches can uncover inefficiencies, bottlenecks, and deviations within processes, empowering analysts to make well-informed decisions and formulate hypotheses about future processes [3].

A fundamental problem studied in process mining is *process discovery*, which involves constructing process models from event data [3]. The discovered models aim to describe the process that generated the data and can vary in detail and accuracy. The event data used as input often takes the form of an *event log*, a collection of *traces*, each captured as a sequence of executed *actions* in a single instance of the process. As the same sequence of actions can be executed multiple times by the process, an event log can contain multiple instances of the same trace.

A plethora of discovery techniques have been proposed, employing a range of options to represent the constructed models. Among these languages, Directly-Follows Graphs (DFGs) stand out for their intuitiveness, and are a preferred choice for practitioners seeking insights [4,25,14]. A DFG is a directed graph in which nodes denote actions and arcs encode "can follow" relations between them. The nodes and arcs of a DFG are annotated with numbers reflecting the frequencies of the actions and "occurs next" dependencies inferred from the data.

In this paper, we present an approach grounded in stochastic grammar inference for constructing a Stochastic Directed Action Graph (SDAG), a special type of DFG defined to capture the likelihood of traces, from an event log. The problem of stochastic grammar inference from a language consists of learning a grammar representing the strings of the language and their probabilities, which indicate their importance in the language [18]. Hence, discovering a process model from an event log is akin to grammar inference, where traces of actions in the event log can be seen as words in the language strings. There are several reasons why one might choose to use stochastic inference for process discovery. First, noisy traces are, in general, infrequent and can thus be identified and suppressed during inference, noting that noise is intrinsic to event data and poses challenges to discovery [15]. Second, stochastic grammars can be used to predict the next trace or deduce the probability of the next action given an observed sequence of actions, information which can inform process simulations [27], decision-making by analysts [2], and future process design [3]. Third, grammar inference is performed based on positive example strings, aiming to: (i) learn the input examples; (ii) favor simpler explanations of the strings, known as Occam's razor or the parsimony principle; and to (iii) generalize to *all* positive examples of the target unknown language [18]. These aims naturally coincide with the goals of process discovery to construct models that: (i) are fitting and precise; (ii) simple; and that (iii) generalize to *all* of the traces the (unknown) process can support [10].

We use *ALERGIA* to perform stochastic grammar inference. *ALERGIA* identifies any stochastic regular language from positive example strings in the limit with probability one [13], with a runtime bounded by a cubic polynomial in the number of input strings. In practice the runtime grows only linearly with the size of the sample set [13,18]. When performing process discovery, to support process exploration at different abstraction levels [30], one is often interested in creating a range of models of various sizes. In general, the problem of determining whether there is a representation of a language of a given size is NP-complete [16]. To control the level of detail in the models it constructs, *ALERGIA* makes use of two parameters. A key contribution in this paper is a genetic algorithm that evolves an initial random population toward parameter pairs that result in better models. Even though SDAGs (unlike DFGs) can have multiple nodes that refer to the same action, we were able to discover SDAGs that are both smaller than the DFGs constructed by a state-of-the-art discovery algorithm and also yield more faithful encodings.

Specifically, we contribute:

1. The first formal semantics of SDAGs (and DFGs) grounded in stochastic languages;
2. A Genetic Algorithm for Stochastic Process Discovery (*GASPD*) that discovers a family of SDAGs from an input event log;

3. A heuristic for focusing genetic mutations to areas likely to accelerate convergence, resulting in SDAGs of superior quality; and
4. An evaluation of *GASPD* over real-life event logs that both demonstrates its benefits and also suggests future improvements.

The remainder of this paper is structured as follows. The next section discusses related work. Section 3 introduces basic notions required to understand the subsequent sections. Then, Section 4 presents SDAGs and their formal semantics. Section 5 proposes our approach for discovering SDAGs from event logs, while Section 6 discusses the results of an evaluation of this approach over real-world datasets using our open-source implementation. The paper concludes with final remarks and discussions in Section 7.

2 Related Work

ALERGIA, introduced by Carrasco and Oncina [13], and its variant, *Minimum Divergence Inference* (*MDI*) by Thollard et al. [34], are state-merging algorithms for learning *stochastic deterministic finite automata* (SDFA) from positive examples. *MDI* extends *ALERGIA* with different heuristics and compatibility tests during state merging. Stolcke and Omohundro [32] proposed *Bayesian Model Merging* (*BMM*), which deduces a model through structure merging guided by posterior probabilities.

Work by Herbst [17], inspired by *BMM*, was the first application of grammatical inference for stochastic process discovery. The approach consists of two routines: model merging and model splitting. The former generalizes the most specific model by merging processes, using log-likelihood as a heuristic, while the latter refines a general model through iterative splits. The resulting models are then converted to ADONIS, permitting concurrent behavior.

Recent research in stochastic process mining resulted in several advancements. Rogge-Solti et al. [31] proposed a technique that lays stochastic performance data over given non-stochastic Petri nets. Improvements of the algorithm by Burke et al. [11] introduce five methods to estimate transition probabilities in Petri nets. An approach developed by Mannhardt et al. [26] discovers data dependencies between Petri net transitions, and Leemans et al. [23] extend the approach to capture stochastic long-dependencies triggered by the earlier actions in processes. To assess the quality of stochastic process models, several quantification techniques have been developed using the Earth Mover's Distance [20], entropy-based conformance checking [24], and the Minimum Description Length principle [29,6].

Directly-Follows Graphs emerged as an alternative modeling notation to deterministic automata. van der Aalst et al. [5] laid the foundation for process discovery by defining the directly-follows relation within a workflow, capturing the inherent dependencies among activities. Algorithms like α-*Miner* [5], *Flexible Heuristics Miner* [35], and *Fodina* [9] discover and map these relations onto Petri nets or BPMN models.

Directly-Follows visual Miner (*DFvM*) discovers DFGs, aiming to visually represent the direct dependencies between actions in the input log. Designed by Leemans et al. [25], *DFvM* also filters out less frequent relations, focusing on significant and regular behaviors. *DFvM* consistently constructs high-quality small-sized models com-

parable to models produced by top software vendors in the field [29], and is the method we use as a baseline in the experiments described in Section 6.

Chapela-Campa et al. [14] proposed a Directly-Follows Graphs filter technique to enhance the understandability of DFGs. The technique formulates the simplification task as an optimization problem, aiming to identify a sound spanning subgraph that minimizes the number of edges while maximizing the sum of edge frequencies.

Widely recognized algorithms like *Inductive Miner* (*IM*) by Leemans et al. [21] and *Split Miner* (*SM*) by Augusto et al. [7] discover DFGs as an intermediate step, mapping them to well-defined modeling notations. *IM* discovers block-structured workflow nets using process trees by computing log cuts based on identified dominant operators such as exclusive choice, sequence, parallel, and loop within the directly-follows graph. Building upon this approach, *Inductive Miner-directly follows* (*IMd*) [22] handles scalability by employing a single-pass directly-follows graph approach for incomplete logs, and those with infrequent behavior. *SM* transforms logs into graphical Directly-Follows Graphs. The algorithm detects concurrency, prunes the graph, and models it in BPMN, balancing precision and fitness and keeping the model complexity low.

The adaptive metaheuristic framework [8] optimizes the accuracy of DFG-based process discovery using three strategies that guide exploring the solution space of discoverable DFGs. The framework iteratively refines DFGs through metaheuristics, evaluates their performance using an objective function, and selects optimal process models.

3 Preliminaries

An SDFA is a representation of the traces of a process and their likelihoods.

Definition 3.1 (Stochastic deterministic finite automata)
A *stochastic deterministic finite automaton* (SDFA) is a tuple $(S, \Lambda, \delta, p, s_0)$, where S is a finite set of *states*, Λ is a finite set of *actions*, $\delta : S \times \Lambda \to S$ is a *transition function*, $p : S \times \Lambda \to [0, 1]$ is a *transition probability function*, and $s_0 \in S$ is the *initial state*, such that $\forall\, s \in S : (\sum_{\lambda \in \Lambda} p(s, \lambda) \leq 1.0)$. ⌟

A *trace* is a sequence $t \in \Lambda^*$. We use ϵ to denote the empty trace. Given two traces t_1 and t_2, their concatenation $t_1 \circ t_2$ is obtained by joining t_1 and t_2 consecutively; for example $\epsilon \circ \epsilon = \epsilon$, $\epsilon \circ \langle \mathtt{b},\mathtt{b} \rangle = \langle \mathtt{b},\mathtt{b} \rangle$, and $\langle \mathtt{b},\mathtt{b} \rangle \circ \langle \mathtt{b},\mathtt{d} \rangle = \langle \mathtt{b},\mathtt{b},\mathtt{b},\mathtt{d} \rangle$.

An SDFA $A = (S, \Lambda, \delta, p, s_0)$ encodes *stochastic language* L_A defined using recursive function $\pi_A : S \times \Lambda^* \to [0, 1]$, that is, $L_A(t) = \pi_A(s_0, t)$, $t \in \Lambda^*$, where:

$$\pi_A(s, \epsilon) := 1.0 - \sum_{\lambda \in \Lambda} p(s, \lambda), \text{ and}$$

$$\pi_A(s, \lambda \circ t') := p(s, \lambda)\ \ \pi_A(\delta(s, \lambda), t'), \lambda \in \Lambda, t = \lambda \circ t'.$$

Note that $\pi_A(s, \epsilon)$ denotes the probability of terminating a trace in state $s \in S$. It holds that $\sum_{t \in \Lambda^*} L_A(t) = 1.0$. Figure 1 shows an example SDFA in which $L_A(\langle \mathtt{a},\mathtt{c},\mathtt{e},\mathtt{c} \rangle) = 0.664$, $L_A(\langle \mathtt{a},\mathtt{b},\mathtt{c},\mathtt{e} \rangle) \approx 0.028$, and $L_A(\langle \mathtt{b},\mathtt{b},\mathtt{b},\mathtt{d} \rangle) = 0$. The probability associated with a trace is an indication of its importance in the language.[1]

[1] A common mistake is to confuse this with the probability of the trace being in the language.

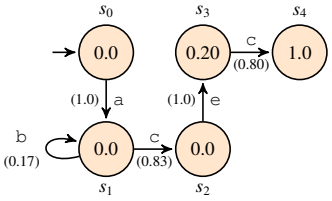


Fig. 1: An SDFA A.

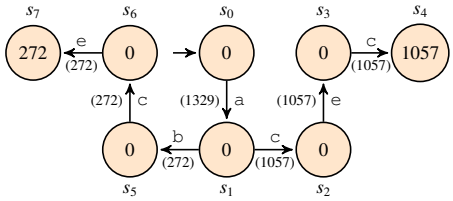


Fig. 2: A PAT T.

An *event log* L is a finite multiset of traces, where each trace encodes a sequence of observed and recorded actions executed in the corresponding process. The multiplicity of t in L, denoted by $n(t, L)$, indicates how frequently it has been observed and recorded; we thus have $|L| = \sum_{t \in L} n(t, L)$. The empirical finite support distribution associated with L, denoted as $\mathcal{P}_L$, is given by $\mathcal{P}_L(t) = n(t, L)/|L|$. For example, suppose that $L = [\langle \mathtt{a}, \mathtt{c}, \mathtt{e}, \mathtt{c} \rangle^{1057}, \langle \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{e} \rangle^{272}, \langle \mathtt{b}, \mathtt{b}, \mathtt{b}, \mathtt{d} \rangle^{164}]$ is an event log. It holds that $|L| = 1493$, $\mathcal{P}_L(\langle \mathtt{a},\mathtt{c},\mathtt{e},\mathtt{c} \rangle) \approx 0.708$, $\mathcal{P}_L(\langle \mathtt{a},\mathtt{b},\mathtt{c},\mathtt{e} \rangle) \approx 0.182$, and $\mathcal{P}_L(\langle \mathtt{b},\mathtt{b},\mathtt{b},\mathtt{d} \rangle) \approx 0.110$.

Entropic relevance relies on the minimum description length principle to measure the number of bits required to compress a trace in an event log using the structure and information about the relative likelihoods of traces described in a model, for instance, an SDFA [6]. Models with lower relevance values to a given log are preferred because they describe the traces and their likelihoods better. For example, the entropic relevance of SDFA A from Figure 1 to event log L is 3.275 bits per trace.[2]

4 Stochastic Directed Action Graphs

We now describe SDAGs, their stochastic semantics, and their relationship to DFGs.

Definition 4.1 (Stochastic directed action graphs) A *stochastic directed action graph* (SDAG) is a tuple $(N, \Lambda, \beta, \gamma, q, i, o)$, where N is a finite set of *nodes*, Λ is a finite set of *actions*, $\beta : N \to \Lambda$ is a *labeling function*, $\gamma \subseteq (N \times N) \cup (\{i\} \times N) \cup (N \times \{o\})$ is the *flow relation*, $q : \gamma \to [0, 1]$ is a *flow probability function*, and $i \notin N$ and $o \notin N$ are the *input* node and the *output* node, such that $\forall\, n \in N \cup \{i\} : (\sum_{m \in \{k \in N \cup \{o\} \mid (n,k) \in \gamma\}} q(n, m) = 1)$. ⌟

An *execution* of an SDAG is a finite sequence of its nodes beginning with i and ending with o, such that for every two consecutive nodes x and y in the sequence there is an arc $(x, y) \in \gamma$. A *trace* of an SDAG is a sequence of actions such that there is an execution that *confirms* the trace in which the nodes, excluding the input and output nodes, are the actions of the trace in the order they appear. Figure 3 shows example SDAG G. The sequence of nodes $\langle i,n_1,n_3,n_5,n_4,o \rangle$ is an execution of G that confirms trace $\langle \mathtt{a},\mathtt{c},\mathtt{e},\mathtt{c} \rangle$.

An SDAG G encodes stochastic language L_G such that for a trace t of G it holds that $L_G(t)$ is equal to the sum of probabilities of all the executions of G that confirm t, where the probability of an execution is equal to the product of the probabilities, as per function q, of all the arcs, as per γ, defined by all pairs of consecutive nodes in the execution. In addition, if $t \in \Lambda^*$ is not a trace of G, it holds that $L_G(t) = 0$.

[2] We use the uniform background coding model throughout this work [6].

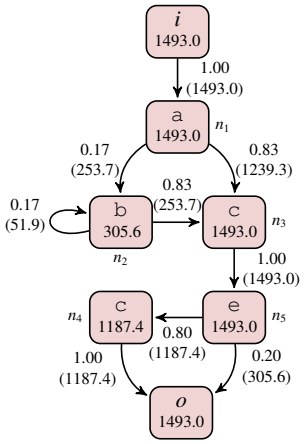


Fig. 3: SDAG G.

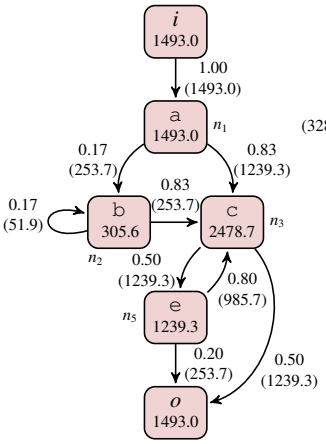


Fig. 4: SDAG G'.

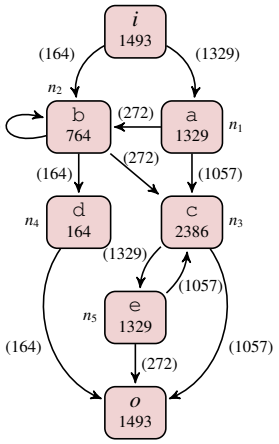


Fig. 5: DFG.

Given an SDFA, one can obtain its corresponding SDAG.

Definition 4.2 (SDAG of SDFA)
Let A be an SDFA (S,Λ,δ,p,s_0). Then, $(N,\Lambda,\beta,\gamma,q,i,o)$, where it holds that:
- $N = \delta$, $i \notin N$, $o \notin N$,
- $\beta = \{((x,\lambda,y),\lambda) \mid (x,\lambda,y) \in \delta\}$,
- $q = \{((x,\lambda_1,y),(y,\lambda_2,z),p(y,\lambda_2)) \mid (x,\lambda_1,y) \in \delta \wedge (y,\lambda_2,z) \in \delta\} \cup$
  $\{(i,(s_0,\lambda,y),p(s_0,\lambda)) \mid (s_0,\lambda,y) \in \delta\} \cup$
  $\{((x,\lambda,y),o,1.0 - \sum_{\mu\in\Lambda} p(y,\mu)) \mid (x,\lambda,y) \in \delta \wedge \sum_{\mu\in\Lambda} p(y,\mu) < 1.0\} \cup$
  $\{(i,o,1.0 - \sum_{\mu\in\Lambda} p(s_0,\mu)) \mid \sum_{\mu\in\Lambda} p(s_0,\mu) < 1.0\}$, and
- $\gamma = \{(x,y) \mid (x,y,z) \in q \wedge z \in [0,1]\}$,

is the *SDAG of* A, denoted by $SDAG(A)$. ⌟

If A is an SDFA, then $SDAG(A)$ is *sound* [1] by construction, as every node of $SDAG(A)$ is on a directed walk from the input node to the output node while the directed walks in $SDAG(A)$ define all and only its executions. Hence, one can reach the output node from every node of $SDAG(A)$ (option to complete property), once a directed walk, and thus the corresponding execution, reaches the output node, it completes (proper completion), and the output node is the only deadlock in $SDAG(A)$ (no dead actions).

In addition, an SDAG of an SDFA has a special structure; that is, it is deterministic.

Definition 4.3 (Deterministic SDAGs) An SDAG $(N, \Lambda, \beta, \gamma, q, i, o)$ is *deterministic* if and only if for every two arcs that start at the same node and lead to two distinct nodes it holds that the labels of the nodes are different, that is, it holds that: $\forall\, n \in N \cup \{i\}\, \forall\, n_1, n_2 \in N : (((n,n_1) \in \gamma \wedge (n,n_2) \in \gamma \wedge n_1 \neq n_2) \Rightarrow \beta(n_1) \neq \beta(n_2))$. ⌟

In a deterministic SDAG, every distinct arc originating at a node must lead to a node with a different action. Consequently, every trace of a deterministic SDAG has only one execution that confirms it, significantly simplifying the computation of the stochastic language of the action graph. The SDAG of an SDFA is deterministic.

Lemma 4.1 (Deterministic SDAGs) *If A is a SDFA then $SDAG(A)$ is deterministic.* ⌟

Lemma 4.1 holds by construction of Definition 4.2. The SDAG G from Figure 3 is an SDAG of SDFA A in Figure 1, i.e., it holds that $G = SDAG(A)$. As G is deterministic, it holds that $L_G(\langle \mathtt{a},\mathtt{c},\mathtt{e},\mathtt{c}\rangle) = 0.664$, which is equal to the product of the probabilities on all the arcs of the corresponding execution $\langle i,n_1,n_3,n_5,n_4,o\rangle$.

Definition 4.4 (SFA of SDAG) Let G be an SDAG $(N, \Lambda, \beta, \gamma, q, i, o)$. Then, $(S, \Lambda, \delta, p, s_0)$, where it holds that $S = N \cup \{i\}$, $\delta = \{(x, \lambda, y) \in (N \cup \{i\}) \times \Lambda \times N \mid (x, y) \in \gamma \wedge \lambda = \beta(y)\}$, $p = \{(x, \lambda, \sum_{\beta(y)=\lambda, y \in N} q(x,y)) \mid x \in N \cup \{i\} \wedge \lambda \in \Lambda \wedge \exists z \in N : (x, \lambda, z) \in \delta\}$, and $s_o = i$, is the *stochastic finite automaton* (SFA) of G, denoted by $SFA(G)$. ⌟

Definition 4.4 generalizes Definition III.5 from [29] to account for the fact that an SDAG can have multiple nodes that refer to the same action.

Note that $SFA(G)$ is indeed an SDFA if G is deterministic.

Lemma 4.2 (Determinism) *If G is a deterministic SDAG then $SFA(G)$ is an SDFA.* ⌟

Lemma 4.2 holds by construction of Definition 4.4. The SDAG of an SDFA and the SDFA of a deterministic SDAG have the same stochastic languages.

Lemma 4.3 (Equivalence) *Let A be an SDFA and let G be an SDAG.*
Then, it holds that (i) $L_A = L_{SDAG(A)}$, and (ii) $L_G = L_{SFA(G)}$. ⌟

The stochastic language of the SDFA of a deterministic SDAG defines the stochastic semantics of the SDAG. In turn, the stochastic semantics of the SDAG of an SDFA is specified by the stochastic language of the SDFA.

In a DFG discovered by a conventional discovery technique, every action has only a single corresponding node. Every SDAG with a pair of distinct nodes that refer to the same action λ can be transformed to an SDAG in which those two nodes are removed, a fresh node for λ is added, all the incoming (outgoing) arcs of the removed nodes get rerouted to reach (originate at) the fresh node, and the probabilities on the outgoing arcs of the fresh node are normalized. Repeated application of this transformation until no further reductions are feasible yields an SDAG that is a DFG.[3] It is straightforward to show that different maximal sequences of feasible transformations do indeed lead to the same resulting DFG. Figure 4 shows the DFG G' obtained in this way from SDAG G in Figure 3; we denote this relationship between the graphs by $G' = DFG(G)$.

An SDAG can be annotated with frequencies of actions and flows, thereby providing information on the rate at which the corresponding concepts arise in the event log from which the SDAG was constructed. Given a number n of cases that should "flow" through the graph, frequencies can be derived from the probabilities by solving a system of equations comprising two types. Each node of the graph defines a *conservation* equation that requires that the sum of frequencies on the incoming arcs equal the sum of the frequencies on the outgoing arcs, except for the input (output) node, for which the sum of frequencies on the outgoing (incoming) arcs is equal to n. Finally, each arc emanating from a node s defines an *arc* equation that specifies that the frequency of the arc is equal to its probability times the sum of frequencies of all the incoming arcs of s. Such

[3] Noting that in a DFG nodes and arcs are annotated with occurrence frequencies.

Table 1: A system of equations used to obtain frequencies of nodes and arcs in Figure 4.

Node	Equation	Arc	Equation
		$\gamma(n_1, n_2)$	$0.17\, f(i, n_1) = f(n_1, n_2)$
i	$1493.0 = f(i, n_1)$	$\gamma(n_1, n_3)$	$0.83\, f(i, n_1) = f(n_1, n_3)$
n_1	$f(i, n_1) = f(n_1, n_2) + f(n_1, n_3)$	$\gamma(n_2, n_2)$	$0.17\, (f(n_1, n_2) + f(n_2, n_2)) = f(n_2, n_2)$
n_2	$f(n_1, n_2) = f(n_2, n_3)$	$\gamma(n_2, n_3)$	$0.83\, (f(n_1, n_2) + f(n_2, n_2)) = f(n_2, n_3)$
n_3	$f(n_1, n_3) + f(n_2, n_3) + f(n_5, n_3) = f(n_3, o) + f(n_3, n_5)$	$\gamma(n_3, n_5)$	$0.50\, (f(n_1, n_3) + f(n_2, n_3) + f(n_5, n_3)) = f(n_3, n_5)$
n_5	$f(n_3, n_5) = f(n_5, n_3) + f(n_5, o)$	$\gamma(n_3, o)$	$0.50\, (f(n_1, n_3) + f(n_2, n_3) + f(n_5, n_3)) = f(n_3, o)$
o	$f(n_3, o) + f(n_5, o) = 1493.0$	$\gamma(n_5, n_3)$	$0.80\, f(n_3, n_5) = f(n_5, n_3)$
		$\gamma(n_5, o)$	$0.20\, f(n_3, n_5) = f(n_5, o)$

a system of equations always has a solution as it contains one unknown per arc and at least as many equations as arcs.

The annotations in Figures 3 and 4 show frequencies for nodes and arcs, obtained from the probabilities by following that process. For instance, the annotations in Figure 4 were obtained by solving the fourteen simultaneous equations in Table 1.[4] Notice that in the figures, all frequencies are rounded to one decimal place. The frequency of a node is derived as the sum of the frequencies of all its incoming (or outgoing) arcs.

The SDAG in Figure 3 is of size 16 and has an *entropic relevance* of 3.267 bits per trace relative to the example event log L from Section 3, with the small difference with the relevance of the SDFA in Figure 1 due to the integer frequencies used in the SDAG. The DFG in Figure 5 constructed from the same log using the *DFvM* algorithm [25] has size 19 and an *entropic relevance* to L of 4.168. That is, the SDAG is smaller and also describes the event log more faithfully than the DFG. Nor does varying the *DFvM* filtering threshold in steps of 0.01 from zero to 1.0 find any outcomes that alter that relativity: 70 of those 100 generated DFGs have a size of 10 and relevance of 6.062; 19 DFGs are of size 14 and relevance of 4.804; and the remaining 11 DFGs (as in Figure 5) have size of 19 and relevance of 4.168. Finally, the relevance of the SDAG in Figure 4 to L is 4.865 bits per trace, illustrating the desirability of permitting duplicate action nodes in the discovered models.

5 Stochastic Process Discovery

Our approach to discovering stochastic process models consists of two fundamental components: a *grammar inference* algorithm coupled with a *genetic optimization* mechanism. We build on *ALERGIA* to construct representations of stochastic language models encoded in the given event data. Complementing that model discovery, we employ *Multi-Objective Genetic Search* to fine-tune the parameters associated with the learning process. This section explains these two fundamental components.

ALERGIA, an instantiation of the *Red-Blue* algorithm, is introduced by Carrasco and Oncina [13] for learning SDFAs from a multiset of strings. The algorithm starts by constructing a *Prefix Acceptor Tree* (PAT) from the multiset of traces, with nodes representing prefixes of traces and edges indicating transitions between the nodes. Each edge is labeled with the frequency of the corresponding trace prefix in the input log. *ALERGIA* then generalizes and compacts that PAT by merging states that exhibit similar behavior. The merging aims to identify sets of states with similar probabilistic dis-

[4] In which $f(x, y)$ is the frequency of arc $(x, y) \in \gamma$.

tributions of outgoing edges, and consolidates each such set into a single state, thereby seeking to create a more concise representation of the underlying language of the model.

Two subsets of nodes are maintained while the compaction process is carried out. The *Red* set initially contains only the root of the prefix tree, while the *Blue* set contains all direct successors of the *Red* set. At each cycle of operation, *ALERGIA* iteratively selects a state from the *Blue* set, taking into account a threshold parameter t that determines the minimum number of strings necessary for a state to be considered for merging. Compatibility for merging is determined by comparing final-state frequencies and outgoing transitions of states in the *Red* and *Blue* sets.

Hoeffding's inequality [19] is a statistical test that bounds the extent to which the mean of a set of observations can deviate from its expected value. This inequality can be used to test if an observed outcome g significantly deviates from a probability value p given a confidence level α and n observations:

$$\left| p - \frac{g}{n} \right| < \sqrt{\frac{1}{2n} \log \frac{2}{\alpha}}\,. \tag{1}$$

ALERGIA approximates that inequality to compare the observed frequencies of pairs of states drawn from the *Red* and *Blue* sets, taking into account final-state frequencies and the frequencies of outgoing transitions:

$$\left| \frac{g_1}{n_1} - \frac{g_2}{n_2} \right| < \omega \cdot \left(\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_2}} \right). \tag{2}$$

In Equation (2), n_1 and n_2 are the frequencies of arriving sequences at the compared states, and g_1 and g_2 are the frequencies of ending sequences at those states. This Hoeffding bound assesses the significance of the difference between the two estimates. It ensures that the difference in frequencies is within a statistically acceptable range, as determined by the value of ω. If the observed frequencies fall within the permitted range, the states are deemed compatible, and *ALERGIA* merges them. Conversely, if the observed differences exceed the bounds, the states are considered incompatible, and the algorithm preserves that difference by converting the *Blue* member of the pair to *Red*. These compatibility checks are not limited to the states but also extend to their respective successors in the tree structure. To merge states, the algorithm redirects transitions and folds subtrees rooted at an identified *Blue* state onto the outgoing edges of the corresponding *Red* state. Based on the merged transitions, the automaton is then updated, with the process iterated until no more mergings can be identified.

As a pre-processing step, we employ a simple filtering technique that selectively retains a certain percentage, as a predefined threshold f, of the most frequent traces from the event log. This filtering reduces the complexity of the discovered model, particularly in scenarios where noise and infrequent traces might detract from the overall view of the process. Varying the filtering threshold f also allows altering the level of detail preserved in the log, allowing alignment with specific analytical goals.

Algorithm 1 provides an overview of *ALERGIA*. The *FILTER* function, called at line 1, filters the input log based on the supplied threshold. The filtered log is then used to construct PAT T, refer to line 2. Figure 2 shows the prefix tree constructed from example log L from Section 3 after applying the filtering using the threshold of $f = 0.89$.

Algorithm 1: *ALERGIA*

Data: A multiset of traces L, parameters $\omega > 0$, $t > 0$, and $f \in [0, 1]$
Result: An SDFA A

1 $L' \leftarrow FILTER(L, f)$;
2 $T \leftarrow PAT(L')$;
3 $Red \leftarrow \{q_0\}$;
4 $Blue \leftarrow \{\delta(q_0, a) : \forall a \in \Sigma\}$;
5 **while** $\exists q_b \in Blue : FREQ(q_b) \geq t$ **do**
6 **if** $\exists q_r \in Red : COMPATIBLE(q_r, q_b, \omega)$ **then**
7 $A \leftarrow MERGE(q_r, q_b)$;
8 $A \leftarrow FOLD(q_r, q_b)$;
9 **else**
10 $Red \leftarrow Red \cup \{q_b\}$;
11 $Blue \leftarrow (Blue - \{q_b\}) \cup \{\delta(q_b, a) : a \in \Sigma$ **and** $\delta(q_b, a) \notin Red\}$;
12 $A \leftarrow CONVERT(T)$;
13 **return** A;

Algorithm 2: *GASPD*

Data: Initial population size n, number of generations *GenLim*, and number of parents k to generate offspring
Result: A Pareto frontier F, captured as a set of parameter triples

1 $g \leftarrow 0$;
2 $P \leftarrow POPULATION(n)$;
3 **while** $g < GenLim$ **do**
4 $F \leftarrow SELECT(P)$;
5 $U \leftarrow CROSSOVER\text{-}MUTATION(F, k)$;
6 $P \leftarrow REPLACE\text{-}ELITE(U, F)$;
7 $g \leftarrow g + 1$;
8 **return** F;

At line 5, the *FREQ* function computes the frequency of arriving at state q_b to check if the state is reached sufficiently frequently, as per the threshold t. The compatibility test between states q_r and q_b is performed by the *COMPATIBLE* function at line 6. The *MERGE* function is then called at line 7 to merge two states and, subsequently, the *FOLD* function folds the tree. Finally, the *CONVERT* function maps T to its corresponding automaton A. For a detailed description of the algorithm, refer to [13]. The SDFA A in Figure 1 is discovered from the example event log by *ALERGIA* using parameters $\omega = 1$, $t = 1$, and $f = 0.89$.

The initialization phase sets the foundation for the genetic algorithm. At line 2 of Algorithm 2, the *POPULATION* function creates a population of n seed solutions $P = \{p_1, p_2, \ldots, p_n\}$, where $p_i = (\omega_i, t_i, f_i)$, $i \in [1 \,..\, n]$, is a parameter triple. The values of these parameters are independently and randomly generated within specified bounds, with each triple determining a model using Algorithm 1. A large initial population enhances exploration but increases the computational cost. Conversely, a small initial population is computationally more efficient but may compromise the ability to find good solutions.

The quality of each solution p_i is then assessed to determine how well it performs with respect to the objective functions. In Algorithm 2, this is done in function *SELECT* at line 4, which applies *ALERGIA* to obtain a process model as a function of each parameter triple in the current population P. This gives rise to a set of models, where each model has an associated size and an *entropic relevance* score computed using *Entropia* [28]. The selection of relevance as a quality metric stems from its ability to rapidly score models, a feature that harmonizes effectively with the genetic framework.

As part of each selection phase, the individuals from the current population that form the *Pareto frontier* F are identified, noting the individuals that are not dominated by other solutions in terms of the two objectives, with Pareto efficient points leading to models having small *size* and *entropic relevance*. Each such point represents a unique trade-off between the objectives, providing decision-makers with a set of alternative options. One such frontier is associated with each *generation*, with the generations counted in Algorithm 2 by the variable g. While we use *ALERGIA*, *size*, and *entropic relevance*, the *GASPD* procedure is not tied to a specific discovery algorithm or quality measure.

In function *CROSSOVER-MUTATION* at line 5 of Algorithm 2, each generation is constructed from the previous one by applying crossover and mutation operations to create a set of offspring, new individuals that (with luck) inherit beneficial traits from previous generations. The *crossover* operation mimics natural genetic recombination, combining information from selected parents, with both single- and double-point crossover techniques employed in our approach. Specifically, to produce offspring, two parents $(\omega_1,\ t_1,\ f_1)$ and $(\omega_2,\ t_2,\ f_2)$ are selected from frontier F. In the single-point crossover, one crossover position in the parent triples is selected. Then, the parameters at and to the left of the crossover point in both parents remain unchanged, while the parameters to the right of the crossover point are swapped between the parents. Hence, six offspring are produced, namely $(\omega_1,\ t_2,\ f_2)$ and $(\omega_2,\ t_1,\ f_1)$ at position one, $(\omega_1,\ t_1,\ f_2)$ and $(\omega_2,\ t_2,\ f_1)$ at position two, and $(\omega_1,\ t_1,\ f_1)$ and $(\omega_2,\ t_2,\ f_2)$ at position three. In the double-point crossover, two additional offspring $(\omega_1,\ t_2,\ f_1)$ and $(\omega_2,\ t_1,\ f_2)$ are produced by swapping parameters of the parents twice, once after the first crossover point and then after the second crossover point. *GASPD* includes all such offspring stemming from all pairs in k randomly selected parents from front F (or $|F|$ selected parents, if $|F| < k$) when generating the set of offspring U.

The *mutation* component of the *CROSSOVER-MUTATION* function then alters the "genetic makeup" of new individuals in U by randomly modifying their three defining parameters, restricted to certain predefined bounds. The random nature of these mutations ensures that the algorithm does not converge prematurely to a limited subset of solutions and allows for the exploration of a broader range of possibilities, crucial when the global optimum may not be immediately apparent. For each individual (ω, t, f) in U, its mutated twin defined as $(\omega + \Delta_\omega, t + \Delta_t, f + \Delta_f)$, where Δ_ω, Δ_t, and Δ_f are random positive or negative mutation values within specified bounds, is created and added to U.

Our approach here (implemented in the *REPLACE-ELITE* function at line 6 of Algorithm 2) is that each generation retains the items that were on the Pareto frontier of any previous generation, and then adds any new parameter triples that establish new points on the frontier. That is, we always preserve solutions that, at some stage, have appeared promising, and seek to add any new solutions that outperform them.

The SDFAs constructed by *ALERGIA* using the parameters discovered by *GASPD* can be translated to sound SDAGs using the principles laid out in Definition 4.2.

6 Evaluation

We have implemented *GASPD*[5] and conducted experiments using twelve publicly available real-world event logs shared by the IEEE Task Force on Process Mining[6], derived from IT systems executing business processes. These experiments assess the feasibility of using *GASPD* in industrial settings, and compare the quality of its models with the ones constructed by *DFvM* [25]. A wide range of *DFvM* filtering parameters were considered, establishing a reference curve for each log showing the versatility of *DFvM* across the spectrum of possible model sizes. We also explored the effectiveness of the genetic search in guiding the selection of parameters to identify desirable solutions.

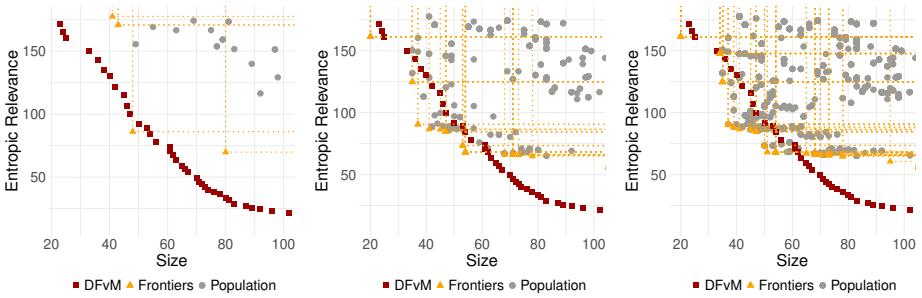
For each of the twelve event logs *GASPD* was seeded with an initial population of 50 parameter triples, each containing random values for ω (from 0 to 15); for t (from 0 to the most frequent PAT branch); and for f (from 0 to 1). The genetic search was then iterated for 50 generations, with mutation achieved by random adjustments to parameter values within the same defined ranges.

Figures 6a to 6c show model relevance scores as a function of model size, taking three snapshots during the course of *GASPD* when executed on the BPIC17 log. After one generation the models are a mixed bag, a result of the random starting point; but with broad coverage of the search space achieved, and already with individuals (yellow dots) identified that outperform the baseline set by the red *DFvM* frontier. By the tenth generation, mutation and breeding have taken the population toward improved performance, with many solutions now below the previous frontier, and convergence towards promising regions in the parameter space. Then, by generation 50, the situation has stabilized, with additional individuals identified that lead to attractive models that outperform the *DFvM* frontier used here as a reference.

Figure 7 shows *GASPD* performance at the 50 generation point for three other logs. In each case *GASPD* constructs models that outperform the *DFvM* ones over at least some fraction of the range of sizes being considered, noting that smaller and more accurate models are preferable for human analysis. Table 2 then summarizes *GASPD* when applied to all twelve event logs, further supporting our contention that *GASPD* finds useful new models. In the first seven rows we focus on human-scale models of up to 100 nodes/edges, and then relax that to a limit of size 1000 for the BPIC15 tasks, which tend to give rise to more complex models. As can be seen, in ten of the twelve cases *GASPD* generates models of better relevance for at least part of the size range, with *DFvM* tending to discover models of better relevance for large sizes. In future work, it would be interesting to study if grammar inference techniques can be useful in discovering large, accurate models.

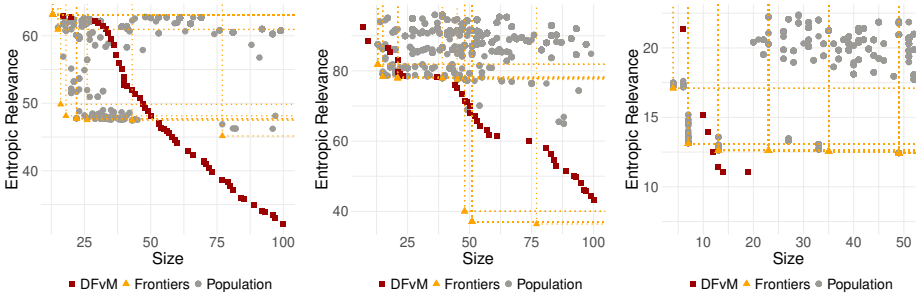
As already described, *GASPD* employs evolutionary search as part of the discovery algorithm. To focus the search onto good candidates, the individuals in each generation P are split into two classes: “good”, and “bad”. An individual is “good” if it has been

[5] https://github.com/jbpt/codebase/tree/master/jbpt-pm/gaspd
[6] https://www.tf-pm.org/resources/logs



(a) Generation 1 (b) Generation 10 (c) Generation 50

Fig. 6: Size and relevance of SDAG models discovered by *GASPD* from the BPIC17 log.



(a) Sepsis (b) BPIC12 (c) BPIC13I

Fig. 7: Size and relevance of SDAG models discovered by *GASPD* after 50 generations.

Table 2: Relative performance of SDAG models discovered by *GASPD* after 50 generations, including the size range of interest, the size interval(s) of *GASPD* superiority, min/max size, and min/max relevance.

Event log	Size interval From	Size interval To	Size interval of superior performance	Size within interval Min	Size within interval Max	Entropic relevance Min	Entropic relevance Max
Sepsis	0	100	(0 .. 53)	13	77	45.16	63.10
BPIC12	0	100	[13 .. 44) ∪ [48 .. 100]	13	77	36.39	81.84
BPIC17	0	100	(0 .. 25) ∪ [34 .. 62)	20	95	60.56	161.25
RTFM	0	100	[53 .. 100]	13	69	2.71	9.07
BPIC13OP	0	100	[4 .. 6) ∪ [8 .. 9)	3	80	5.06	7.80
BPIC13CP	0	100	(0 .. 9)	4	79	6.26	9.78
BPIC13I	0	100	(0 .. 12)	4	89	12.33	17.10
BPIC15-1	0	1000	[37 .. 44) ∪ [59 .. 122)	37	633	380.01	384.21
BPIC15-2	0	1000	[473 .. 641) ∪ [697 .. 965)	149	946	428.00	469.94
BPIC15-3	0	1000	(0 .. 27) ∪ [95 .. 103)	11	958	298.01	370.63
BPIC15-4	0	1000	∅	475	475	386.93	386.93
BPIC15-5	0	1000	∅	695	695	445.41	445.41

on the Pareto frontier in any previous generation; and it is “bad” if has never been part of any Pareto frontier. In Algorithm 2 each generation is derived solely from the “good” individuals of the previous population, with the aim of iterating towards better solutions. Only new individuals that are also “good” are then retained into the next population.

To verify the usefulness of that heuristic, we also experimented with breeding from “bad” parents, recording at each generation the fractions of “good” offspring from pairs

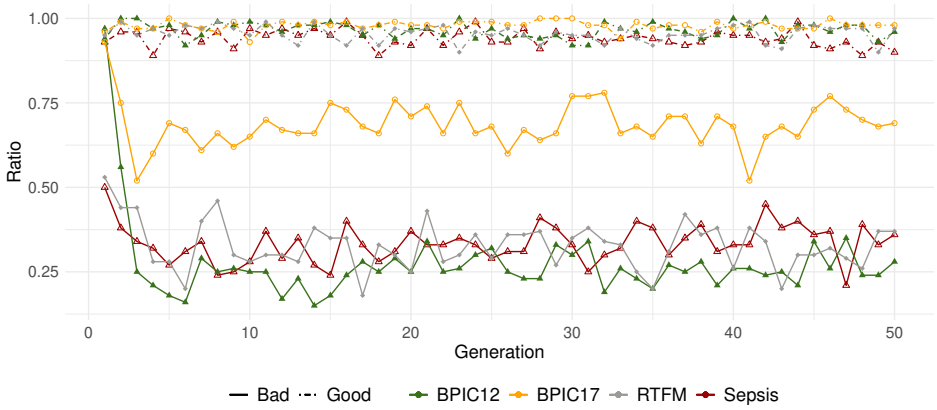


Fig. 8: The fraction of "good" individuals arising as offspring from pairs of "good" parents and pairs of "bad" parents, traced through 50 generations, for each of four logs.

of "bad" parents, and then likewise the fraction arising from pairs of "good" parents. Figure 8 shows the result, with the four dotted lines at the top the "good parents" success rate, and the four solid lower lines the "bad parents" rate. As can be seen, the four dotted lines comprehensively outperform their solid equivalents, and good parents are much more likely to lead to interesting offspring than are bad parents. Taking a paired t-test between the 50 "dotted" points and the 50 "solid" points for the four logs gives $p < 10^{-10}$ in all four cases, confirming that filtering the population at each generation to only contain good parents is a highly beneficial strategy.

7 Discussion and Conclusion

We have presented *GASPD*, an algorithm for discovering sound Stochastic Directed Action Graphs, a variant of the DFGs often used in commercial process mining applications. The technique is grounded in grammar inference over sequences of actions, as recorded in event logs; and uses a bespoke genetic algorithm that ensures fast convergence towards models of practically interesting sizes and accuracy.

GASPD can be used to implement a user-friendly tool for exploring input logs, with interesting models often uncovered even in the first generation of random parameters, and then improved through subsequent generations. Moreover, construction of multiple such models is easily done in parallel, meaning that as the interesting models become progressively available, they can be presented to the user via a dynamic slider interface [30], ordered by size. This conjectured interface can thus present the so-far discovered Pareto-best models, while generations are extended and new models are constructed in the background, further feeding the slider. A second slider can be introduced to navigate over all the computed generations. For instance, selecting a specific generation in this slider can load all the Pareto optimal models obtained after that generation in the other slider. Such controls would support interactive exploration of the improvement in model quality observed through generations.

The exploration of alternative grammatical inference techniques stands out as a promising avenue for enhancing the quality of the discovered models. Replacing *ALERGIA* with other grammar inference methods in the genetic framework can reveal their ultimate effectiveness in generating process models from event logs. *ALERGIA* appears as a strong candidate for initiating the quest due to its acknowledged performance characteristics over a wide variety of real-world languages [18]. Additionally, there is a prospect for improvement by introducing pruning strategies over prefix acceptor trees before merging states and redesigning the merging rules. These enhancements can streamline the algorithm's convergence process, ensuring more efficient solution space exploration, resulting in more accurate and concise process models and allowing the discovery of superior models of larger sizes. Finally, new ideas to guide the genetic exploration of the parameter space of grammar inference can be explored. Some initial ideas include using simulated annealing and particle swarm optimization principles to escape parameter subspaces of local optimal models.

The evaluation in Section 6 made use of entropic relevance and model size to identify Pareto-optimal models. Entropic relevance ensures models that better describe the frequencies of log traces are prioritized, and strikes a balance between conventional precision and recall quality measures in process mining [6]; and model size is a standard measure of simplicity for DFGs [4]. But other measurement combinations could also be considered, and would lead to different concrete measurements and, consequently, might result in different Pareto-optimal models. Nevertheless it seems probable that measures that assess the same broad phenomena as entropic relevance and size will result in similar conclusions to those achieved here – that *GASPD* provides the ability to realize interesting and useful process models not found by other current approaches, making it an important development for process mining.

References

1. van der Aalst, W.: Verification of workflow nets. In: ICATPN. LNCS, vol. 1248 (1997)
2. van der Aalst, W.: Using process mining to bridge the gap between BI and BPM. Computer **44**(12) (2011)
3. van der Aalst, W.: Process Mining—Data Science in Action. Springer, 2nd edn. (2016)
4. van der Aalst, W.: A practitioner's guide to process mining: Limitations of the directly-follows graph. Procedia Comput. Sci. **164** (2019)
5. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9) (2004)
6. Alkhammash, H., Polyvyanyy, A., Moffat, A., García-Bañuelos, L.: Entropic relevance: A mechanism for measuring stochastic process models discovered from event data. Inf. Syst. **107** (2022)
7. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: Automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2) (2019)
8. Augusto, A., Dumas, M., La Rosa, M., Leemans, S., vanden Broucke, S.: Optimization framework for DFG-based automated process discovery approaches. Softw. Syst. Model. **20**(4) (2021)
9. vanden Broucke, S., De Weerdt, J.: Fodina: A robust and flexible heuristic process discovery technique. Decis. Support Syst. **100** (2017)

10. Buijs, J., van Dongen, B., van der Aalst, W.: Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. Int. J. Coop. Inf. Syst. **23**(01) (2014)
11. Burke, A., Leemans, S., Wynn, M.T.: Stochastic process discovery by weight estimation. In: ICPM-WS. LNBIP, vol. 406 (2020)
12. Burke, A., Leemans, S., Wynn, M.T.: Discovering stochastic process models by reduction and abstraction. In: Petri Nets. LNCS, vol. 12734 (2021)
13. Carrasco, R., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: ICGI. LNCS, vol. 862 (1994)
14. Chapela-Campa, D., Dumas, M., Mucientes, M., Lama, M.: Efficient edge filtering of directly-follows graphs for process mining. Inf. Sci. **610** (2022)
15. Cheng, H.J., Kumar, A.: Process mining on noisy logs: Can log sanitization help to improve performance? Decis. Support Syst. **79** (2015)
16. Gold, E.M.: Complexity of automaton identification from given data. Inf. Control. **37**(3) (1978)
17. Herbst, J.: A machine learning approach to workflow management. In: ECML. LNCS, vol. 1810 (2000)
18. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, 1st edn. (2010)
19. Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301) (1963)
20. Leemans, S., van der Aalst, W., Brockhoff, T., Polyvyanyy, A.: Stochastic process mining: Earth movers' stochastic conformance. Inf. Syst. **102** (2021)
21. Leemans, S., Fahland, D., van der Aalst, W.: Discovering block-structured process models from event logs: A constructive approach. In: Petri Nets. LNCS, vol. 7927 (2013)
22. Leemans, S., Fahland, D., van der Aalst, W.: Scalable process discovery and conformance checking. Softw. Syst. Model. **17**(2) (2018)
23. Leemans, S., Mannel, L.L., Sidorova, N.: Significant stochastic dependencies in process models. Inf. Syst. **118** (2023)
24. Leemans, S., Polyvyanyy, A.: Stochastic-aware conformance checking: An entropy-based approach. In: CAiSE. LNCS, vol. 12127 (2020)
25. Leemans, S., Poppe, E., Wynn, M.T.: Directly follows-based process mining: Exploration & a case study. In: ICPM (2019)
26. Mannhardt, F., Leemans, S., Schwanen, C., de Leoni, M.: Modelling data-aware stochastic processes - discovery and conformance checking. In: Petri Nets. LNCS, vol. 13929 (2023)
27. Meneghello, F., Di Francescomarino, C., Ghidini, C.: Runtime integration of machine learning and simulation for business processes. In: ICPM (2023)
28. Polyvyanyy, A., Alkhammash, H., Di Ciccio, C., García-Bañuelos, L., Kalenkova, A., Leemans, S., Mendling, J., Moffat, A., Weidlich, M.: Entropia: A family of entropy-based conformance checking measures for process mining. In: ICPM. CEUR-WS Proc., vol. 2703 (2020)
29. Polyvyanyy, A., Moffat, A., García-Bañuelos, L.: An entropic relevance measure for stochastic conformance checking in process mining. In: ICPM (2020)
30. Polyvyanyy, A., Smirnov, S., Weske, M.: Process model abstraction: A slider approach. In: ECOC (2008)
31. Rogge-Solti, A., van der Aalst, W., Weske, M.: Discovering stochastic petri nets with arbitrary delay distributions from event logs. In: Bus. Process. Manag.-WS. LNBIP, vol. 171 (2013)
32. Stolcke, A., Omohundro, S.: Best-first model merging for hidden Markov model induction. CoRR **9405017** (1994)

33. Tax, N., Lu, X., Sidorova, N., Fahland, D., van der Aalst, W.: The imprecisions of precision measures in process mining. Inf. Process. Lett. **135** (2018)
34. Thollard, F., Dupont, P., de la Higuera, C.: Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In: ICML (2000)
35. Weijters, T., Ribeiro, J.: Flexible heuristics miner (FHM). In: CIDM (2011)