Discovering Stochastic Process Models By Reduction and Abstraction

Adam Burke[0000−0003−4407−2199], Sander J.J. Leemans[0000−0002−5201−7125], and Moe Thandar Wynn[0000−0002−7205−8821]

Queensland University of Technology, Brisbane, Australia
{at.burke, s.leemans, m.wynn}@qut.edu.au

Abstract. In process mining, extensive data about an organizational process is summarized by a formal mathematical model with well-grounded semantics. In recent years a number of successful algorithms have been developed that output Petri nets, and other related formalisms, from input event logs, as a way of describing process control flows. Such formalisms are inherently constrained when reasoning about the probabilities of the underlying organizational process, as they do not explicitly model probability. Accordingly, this paper introduces a framework for automatically discovering stochastic process models, in the form of Generalized Stochastic Petri Nets. We instantiate this Toothpaste Miner framework and introduce polynomial-time batch and incremental algorithms based on reduction rules. These algorithms do not depend on a preceding control-flow model. We show the algorithms terminate and maintain a deterministic model once found. An implementation and evaluation also demonstrate feasibility.

Keywords: Stochastic Petri nets · process mining · stochastic process discovery · stochastic process mining

1 Introduction

Modelling is a way for us to understand and navigate the world; some thinkers argue it is the core activity of science [41]. Today's world, with its cheap computers and voluminous data, makes new forms and subjects of modelling possible. The last two decades have seen great progress in one form, process mining [1] – the analysis of organizational processes using computational techniques and large event logs. Process-mined models are then used to understand and improve organizations.

Stochastic process models, such as Stochastic Petri Nets [4], are well-established in fields from biology [26] to operations research [42] to describe evolving processes with complex causalities, and relative probabilities. *Stochastic process mining* discovers and analyzes stochastic process models. It is a relatively new area of research which aims to exploit the sophistication of stochastic models to advance our understanding of organizations and their frequent, or infrequent, behaviour. Currently few discovery and conformance techniques exist. Importantly, these existing discovery techniques do not work well for a number of important

real-world cases. They also often depend on control-flow models, limiting the use of stochastic information in the construction of the control-flow itself.

In this paper we introduce a new stochastic process discovery framework, *Toothpaste Miner*, which works with Generalized Stochastic Petri Nets (GSPNs) [25,4] via a data structure targeted at process mining, the Probabilistic Process Tree. Toothpaste Miner does "direct stochastic discovery", i.e., it does not rely on an initial control-flow discovery step, but calculates control-flow and stochastic aspects of the model using a common abstraction. It proceeds by the repeated application of reduction rules. We show polynomial-time computational complexity, termination and deterministic properties of these algorithms. An implementation, in Haskell, and its evaluation, against real-life event logs, shows the technique's practical relevance, and also that it trades off quality against more complex models and longer execution times.

In Sections 2 and 3, below, we discuss related work and foundational concepts. The Toothpaste Miner discovery algorithms and transformation rules are introduced in Section 4, together with the Probabilistic Process Trees formalism. Incremental discovery and noise-management optimisations are discussed in Section 5. The implementation and evaluation are laid out in Section 6, before we conclude in Section 7.

2 Related Work

Important existing work in this area includes that on stochastic process mining, discovering Petri nets, and discovering probabilistic automata.

The stochastic process mining algorithms introduced in Sections 4.2 and 5 are partially region-based. A number of process mining algorithms for region-based control-flow discovery exist [10]. The *Maximal Pattern Mining* algorithm [23] is a region-based algorithm which combines regular expression-like patterns in systematic ways, and helped inspire the loop and concurrency identification rules in Section 4.3. Other sources for rules are Petri net and Stochastic Petri net reductions [40,35] and the Inductive Miner [20], which uses process trees. The Probabilistic Process Trees introduced here extend process trees. (The term "stochastic process tree" is already used to refer to decision trees, e.g. in [13].)

Within process mining, existing stochastic process discovery techniques can be categorized as control-flow dependent, direct, or declarative. For control-flow dependent discovery, one key technique discovers Generally Distributed Stochastic Petri Nets (GDT_SPNs) after alignment-based repair [30,31]. Other techniques output Generalized Stochastic Petri Nets [9], trading some quality for faster execution times, or combine control-flow models using Bayesian inference [16]. Direct discovery techniques exist in the literature, as high level descriptions or algorithms [33,17,2,15] and for structures other than Petri nets. One recent discovery technique shows reduced error percentages by using a Bayesian network with non-classical probability [28]. It is however constrained to exclude loops and concurrency. Another recent technique [29] re-purposes the Direct Follows Graph Miner [21] to obtain a stochastic Direct Follows Model. Discovery of declarative stochastic process models has also seen good progress in recent

years [5,24], though the difficulties of comparing control-flow and declarative models put them beyond the immediate scope of this article.

The problem of discovering probabilistic models from event data has three broad classes of existing solutions validated by empirical trials [37]: Bayesian inference, state merging, and parameter estimation.

Bayesian inference is a method where probability estimates are updated in a specific form of cumulative average in response to the introduction of new evidence. Bayesian inference on its own does not yield a structured and visualizable model, just probability estimates for particular events, so cannot be directly applied to process mining [8]. Probabilities obtained with Gibbs sampling [14] have recently been successfully combined with an input control flow model for stochastic process discovery [16].

State merging is exemplified by the ALERGIA algorithm [12], which discovers Stochastic Finite-State Automata through state merges in cubic time; alternatives such as MDI [36] achieve quadratic time complexity. ALERGIA can still be competitive in real-world trials [37]. Both ALERGIA and MDI construct an internal prefix automaton with weights. This general algorithmic structure is also used by our discovery algorithm in Section 4.2 and merge operators and rules in Section 4.3. We adopt a Petri net-based data structure used in process discovery algorithms, process trees [20],[1, p81], to manage state merges, instead of the prefix trees used in ALERGIA.

The RegPFA framework [8] uses parameter estimation to do process prediction. RegPFA uses an internal model for prediction based on Baum-Welch [3]. It outputs a noise-filtered Petri net model for user consumption, which emphasizes understandability against precision, and elides stochastic information.

To the best of our knowledge, the proposed techniques represent novel solutions for discovering stochastic process models. The framework uses a well-established process formalism (GSPNs) and supports loops and concurrency. Rather than annotating stochastic information after finding a control-flow model, it makes direct use of trace information from the event log to construct the stochastic aspect of the model in concert with the control-flow.

3 Preliminaries

Generalized Stochastic Petri Nets (GSPNs) [25,4] and Stochastic Deterministic Finite Automata [38,39] are well-established formalisms, and good overviews exist [25,4,38,39]. Definitions in this section are based on the process mining and Petri net literature [1, p80],[19]. We use $\mathbb{N}$ for natural numbers, $\mathbb{R}^+$ for positive real numbers, and $\mathbb{B}$ for booleans; $\bullet$ separates quantifiers and predicates.

Event logs. A process consists of various *activities*. Let the set A be an alphabet of activities for a process. A trace is a sequence of activities. Σ^* is the set of all possible traces over A. A *language subseteq*Σ^* is a set of traces. An event log L is a finite multiset of traces collating observations of the underlying process. Let $|L|$ represent the number of traces and $||L||$ the number of activities in the log. A log with ten traces of $\langle a, b\rangle$ and six traces of $\langle b, c\rangle$ is written $[\langle a, b\rangle^{10}, \langle b, c\rangle^{6}]$ following multiset notation in [32].

Definition 1 (Petri nets). *A* Petri net *[1,19] is a tuple* $PN = (P, T, F, M_0)$*, where* P *is a finite set of places,* T *is a finite set of transitions, and* $F \colon (P \times T) \rightarrow (T \times P)$ *is a flow relation. A* marking *is a multiset of places* $\subseteq P$ *that indicate a state of the Petri net, with* M_0 *being the initial marking. A transition is* enabled *if every incoming place contains a token. A transition fires by changing the marking of the net to consume incoming tokens and producing tokens for its outgoing places. A net where no further transitions may fire has reached a terminal state and corresponds to a final marking.*

Definition 2 (Generalized Stochastic Petri Net (GSPN)). *A* GSPN *[25,4] is a tuple* $(P, T, F, M_0, W, T_i, T_t)$ *such that* (P, T, F, M_0) *is a Petri net. Weight function* $W \colon T \rightarrow \mathbb{R}^+$ *assigns each transition a weight.* T_i *is a set of immediate transitions and* T_t *a set of timed transitions such that* $T_i \cup T_t = T$ *and* $T_i \cap T_t = \varnothing$*. If multiple transitions* $T_e \subseteq T_i$ *are enabled in a particular marking, the probability of a transition* $t \in T_e$ *firing is given by* $\frac{W(t)}{\Sigma_{t' \in T_e} W(t')}$*. Immediate transitions take priority over timed transitions. A timed transition, if enabled, fires according to an exponentially distributed wait time. Given a set of enabled timed transitions* $T_e \subseteq T_t$*, a transition* t *fires first with probability* $\frac{W(t)}{\Sigma_{t' \in T_e} W(t')}$*.*

Definition 3 (Generalized Stochastic Labelled Petri Net (GSLPN)). *A* GSLPN *[22] is a tuple* $(P, T, F, M_0, W, T_i, T_t, \Sigma, \lambda)$ *where* $(P, T, F, M_0, W, T_i, T_t)$ *is a GSPN.* λ *is a labelling function for transitions* $\lambda \colon T \rightarrow \Sigma \cup \{\tau\}$ *where* $\tau \notin \Sigma$*. When a transition* $t \in T$ *fires, if* $\lambda(t) = a$ *where* $a \neq \tau$*, the activity* a *has executed.* $\lambda(t) = \tau$ *is a* silent transition *where there is no evidence of activity execution. The set of GSLPNs with only immediate transitions is denoted by* $\mathcal{G}$*.*

Definition 4 (Stochastic language). *A stochastic language* $L_\Sigma : \Sigma^* \rightarrow [0, 1]$ *is a function which denotes a probability for each trace such that* $\Sigma_{t \in \Sigma^*} L_\Sigma(t) = 1$*.*

Definition 5 (Stochastic Deterministic Finite Automaton). *A Stochastic Deterministic Finite Automaton (SDFA) [11] is a tuple* (S, A, σ, p, s_0) *where* S *is a set of states,* A *is an alphabet,* $\sigma : S \times A \rightarrow S$ *is a transition function,* $p \colon S \times A \rightarrow [0, 1]$ *maps state probability, and the initial state is* $s_0 \in S$*.*

SDFAs are a special case of Probabilistic Finite-State Automata (PFAs) [38] and SDFAs are also referred to as Deterministic Probabilistic Finite Automata (DPFA) [38]. All event logs can be represented by SDFAs [19].

4 Process Discovery By Model Reduction

In this section we first introduce *Probabilistic Process Trees* (PPTs), which add relative weights and some alternative operators to the process tree formalism, and can be translated straightforwardly to GSLPNs. We then describe a novel algorithm which uses Probabilistic Process Tree transformations for process model discovery in terms of general rule properties. Lastly, concrete transformation rules for manipulating these trees are introduced.

Table 1: Translation of PPTs to GSLPNs.

Probabilistic Process Tree	GSLPN
$\oplus\colon w$ (root)	I — operator-specific translation of $\oplus\colon w$ — O
$\rightarrow\colon w$ with $x\colon w$, $y\colon w$	$x\colon w$ → $y\colon w$
$\times\colon w$ with $x\colon v_x$, $y\colon v_y$	$x\colon v_x$ ∥ $y\colon v_y$
$\wedge\colon w$ with $x\colon v_x$, $y\colon v_y$	$\tau\colon w$ → ($x\colon v_x$ ∥ $y\colon v_y$) → $\tau\colon w$
$\circlearrowright_p^\rho\colon w$ with $x\colon w$	$\tau\colon w$ → ($\tau\colon 1$ ∥ $x\colon \rho-1$)
$\circlearrowright_n^m\colon w$ with $x\colon w$	$x\colon w$ → ...m times (inclusive)... → $x\colon w$
$a\colon w$	$a\colon w$
$\tau\colon w$	$\tau\colon w$

4.1 Probabilistic Process Trees

A *Probabilistic Process Tree* is view of a GSLPN, and a cousin of the process tree [1, p78] which allows probabilistic choice. As in Section 3, A refers to a finite set of activities with silent activity $\tau \notin A$. We first define trees recursively, followed by operators.

Definition 6 (Probabilistic Process Trees). *A Probabilistic Process Tree (PPT) is a tree of weighted nodes, where each node is denoted by $s\colon w$. The universe of PPTs over activity set A is $\mathcal{U}_\mathcal{A}$:*

1. *A single activity. For $a \in A$, $a\colon w \in \mathcal{U}_\mathcal{A}$.*
2. *A silent activity. For $\tau \notin A$, $\tau : w \in \mathcal{U}_\mathcal{A}$.*
3. *An operator $\oplus$ over one or more child trees. Given $n \geqslant 1$, $U_1, ..., U_n \in \mathcal{U}_\mathcal{A}$, then $\oplus(U_1, ..., U_n)\colon w \in \mathcal{U}_\mathcal{A}$.*

Sub-trees. U_i is a *sub-tree* of its parent $U = \oplus(U_1, ...U_i, ..., U_n) : w$, and of all trees of which U is a sub-tree. Trees are *strictly equal*, $U_x = U_y$, when the structures are isomorphic and each node and sub-tree is equal, including weights. Activity powerset function *exp* gives the set of all activities in a given PPT.

In this paper, we consider operators $\bigoplus = \{\rightarrow, \wedge, \times, \circlearrowleft_p, \circlearrowleft_n\}$. These redefine and extend non-probabilistic process tree operators to include weight semantics.

Sequential operator. The $\rightarrow$ sequential operator executes its children in sequential order. To align weight with execution frequency, the weight of sequence components is the parent's weight. In the remainder of this paper, we may omit the child weights, writing $\rightarrow (x, y) : w$ rather than $\rightarrow (x : w, y : w) : w$.

Choice. The probabilistic choice operator $\times(U_1, ..., U_n)$ chooses one sub-tree $U_i = s_i : w_i$ for execution from its children, with probability $\frac{w_i}{\Sigma_{s_j \,:\, w_j \in \{U_1, ..., U_n\}} w_j}$.

Concurrency. The concurrency operator $\wedge$ indicates parallel composition. Each child must execute once with no constraint on order of execution.

Fixed Loops. The fixed loop operator $\circlearrowleft_n^m (U)$ repeats its child tree m times. As for sequences, the weight of the loop child is the weight of the parent loop.

Probabilistic Loops. The probabilistic loop operator $\circlearrowleft_p^\rho (U)$ executes the child tree with the probability of exiting at each iteration determined by the function $Pr\{\chi = 0\} = \frac{1}{\rho}$ where $\rho \in \mathcal{R}^+ \wedge \rho \geqslant 1$. The weight of the child node is the weight of the parent loop.

$$\forall x, y, w_1, w_2, w_3 \bullet \rightarrow (x : w_1, y : w_2) : w_3 \in \mathcal{U}_\mathcal{A} \implies w_1 = w_2 = w_3$$
$$\forall s_1, .., s_n, w_1, .., w_n \bullet \times(s_1 : w_1, ..., s_n : w_n) : w \implies w = \Sigma_{(s_j, w_j)} w_j$$
$$\forall s_1, .., s_n, w_1, .., w_n \bullet \wedge(s_1 : w_1, ..., s_n : w_n) : w \implies w = \Sigma_{(s_j, w_j)} w_j$$
$$\forall m, x, w_1, w_2 \bullet m \in \mathbb{N} \wedge \circlearrowleft_n^m \ (x : w_1) : w_2 \implies w_1 = w_2$$
$$\forall m, x, w_1, w_2 \bullet \rho \in \mathbb{R}^+ \wedge \circlearrowleft_p^\rho \ (x : w_1) : w_2 \implies w_1 = w_2$$

Size. The size of a PPT is the number of nodes, denoted by $|U_A|$.

Example. One PPT, $pe = \times(\rightarrow (a : 2, b : 2) : 2, \rightarrow (b : 1, a : 1) : 1, \circlearrowleft_p^3 (c : 1) : 1) : 4$, can be seen in Figure 1, and has the stochastic language $\Sigma_E = [\langle a, b\rangle^{\frac{1}{2}}, \langle b, a\rangle^{\frac{1}{4}}, \langle c, c, c\rangle^{\frac{1}{4}}]$.

Translation to Generalized Stochastic Petri Nets. Having defined the syntax, and informally explained the meaning of PPT constructs, we now formally define the semantics. PPTs have equivalent constructs in a Generalized Stochastic Petri Net (GSPN), summarized in Table 1.

Note that in the probabilistic loop $\circlearrowleft_p^\rho$, each iteration is a Bernoulli trial with the number of iterations being a geometric variable. In the GSPN translation, this results

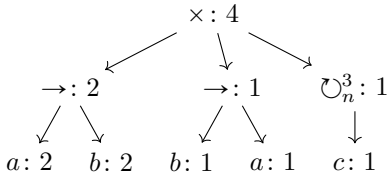


Fig. 1: Example PPT *pe*.

in a transition weight of $\rho - 1$:

$$Pr(\circlearrowright_p \text{ exit}) = \frac{w_{exit}}{\Sigma w_i} \qquad \text{GSPN choice definition}$$
$$= \frac{1}{1 + (\rho - 1)} \qquad = \frac{1}{\rho} \circlearrowright_p \text{ definition}$$

The translation of PPTs to GSPNs, as described in Table 1, shows PPTs are a subset of Probabilistic Finite Automata [38]. Figure 2 gives the translation of example PPT *pe* into a GSPN, with a more sophisticated example shown in Figure 4f.

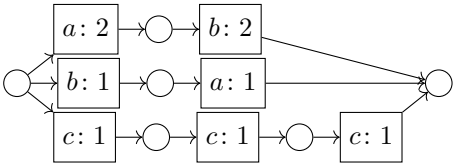


Fig. 2: Translation of *pe* to a GSPN.

4.2 A Discovery Algorithm Framework

The *Toothpaste framework* describes reduction algorithms that "squeeze" an initial trace model into a more summarized and useful form using transformation rules. The framework is illustrated in Figure 3. After introducing component elements, an example instantiation is made in Definition 7.

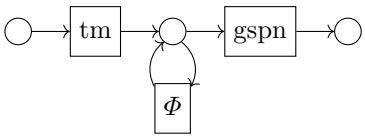


Fig. 3: Toothpaste framework.

Toothpaste miner algorithms first transform an event log into an internal PPT. They then repeatedly transform the PPT by applying transformation rules. These rules reduce, summarize, or restructure the tree towards a desirable form. When desired criteria for an output process model are met, the PPT is translated into a GSLPN as the final output. Criteria may include quality criteria such as fitness or precision thresholds, simplicity, or the preservation of certain critical trace paths in the final model. As the miner proceeds largely by reduction from an initial state which perfectly matches the event log, this allows for fine-grained control over what elements of the initial log are preserved in the final model.

Event logs and discovery. Given an event log $L = [t_1^{i_1}, ..., t_n^{i_n}]$, a trace model PPT is given by $tm : L \to \mathcal{U}_{\mathcal{A}}$, where each trace is converted by $st : \langle A \rangle \to \mathcal{U}_{\mathcal{A}}$.

$$st(\langle a_1, ..., a_m \rangle) = \to (a_1 : i_j, ..., a_m : i_j) : i_j$$
$$st(\langle a \rangle) = a : i_j$$
$$tm(L) = \times(st(t_1), ..., st(t_n)) : |L|$$

For example, $tm(\langle a, b \rangle, \langle c \rangle^3) = \times(\to (a, b) : 1, c : 3) : 4$.

Transformation. In the Toothpaste framework, discovery proceeds by the application of transformation rules to a PPT, yielding progressively improved models. For rule sequences, + is used for concatenation and **ran** for the set of elements in a sequence. We instantiate the framework using *reduction rules*, those which reduce the total number of nodes in the tree, with specific rules in Section 4.3.

Function Φ applies a sequence of transformation rules to a PPT.

$$\Phi \colon \mathcal{U}_{\mathcal{A}} \times \langle \mathcal{U}_{\mathcal{A}} \to \mathcal{U}_{\mathcal{A}} \rangle \to \mathcal{U}_{\mathcal{A}}$$
$$\Phi(pt, \langle\rangle) = pt$$
$$\Phi(pt, \langle r \rangle) = r(pt)$$
$$\Phi(pt, \langle r \rangle + rs) = \Phi(r(pt), rs))$$

Φ_M finds a local reduction minima by applying Φ exhaustively.

$$\Phi_M \colon \mathcal{U}_{\mathcal{A}} \times \langle \mathcal{U}_{\mathcal{A}} \to \mathcal{U}_{\mathcal{A}} \rangle \to \mathcal{U}_{\mathcal{A}}$$
$$\Phi_M(pt, rs) = \begin{cases} \Phi_M(pt', rs) & \text{if } pt \neq pt' \\ pt & \text{otherwise} \end{cases}$$
$$\text{where } pt' = \Phi(pt, rs)$$

Both Φ and Φ_M are guaranteed to terminate when used with reduction rules, as the size of the input tree is monotonically decreasing. Informally, so long as rules are chosen to preserve fidelity to the log, a minimal reducible model is desirable. The degree to which fidelity is desirable can be controlled by which rules are provided to the discovery algorithm. If a given ruleset is not confluent [7, p10], finding a minimal model is not guaranteed and can depend on the sequence in which the rules are applied.

Definition 7 (Toothpaste miner).

Given reduction rule sequence rs and GSLPN translation function $gspn$,

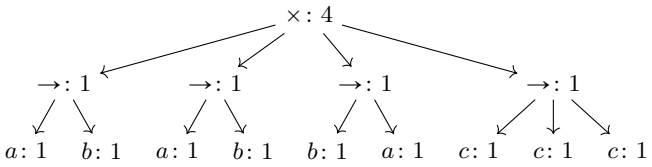
$$dtm \colon L \times \langle \mathcal{U}_{\mathcal{A}} \to \mathcal{U}_{\mathcal{A}} \rangle \to \mathcal{G}$$
$$dtm(L, rs) = gspn \circ \Phi_M(tm(L), rs) \text{ is the direct toothpaste miner.}$$

Example. An example of model discovery is in Figure 4, applying (and previewing) rules from Section 4.3. The trace model in 4a has identical $\langle a, b \rangle$ traces consolidated in model 4b. The repeated c activities are summarized with a fixed loop in 4c. Concurrency of events a and b is identified in 4d, and a probabilistic loop is introduced in 4e. Finally the PPT is translated to a GSPN in 4f.

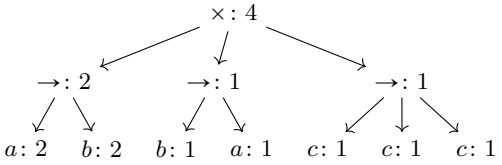
Complexity. The computational complexity of the Φ algorithms depend on the size of the PPT data structure. Function st produces a binary tree with $2|t_i| - 1$ nodes for each trace t_i. The full trace model produced by tm adds a choice node, for a total size (and memory complexity) of $\Sigma(2|t_i| - 1) + 1 = 2||L|| - |L| + 1$ or $O(||L||)$.

The complexity of Φ depends on evaluating each node of the tree with reduction rules. If each sub-tree can be summarized with one traversal, the worst case is comparing each node to each other node, giving $O(\Phi(U_A, R)) = |U_A|^2|R|$ comparisons. Writing U_L for the full PPT trace model produced by tm, this is limited by $(2||L||)^2|R|$ or $O(\Phi(U_L, R)) = O(||L||^2|R|)$.

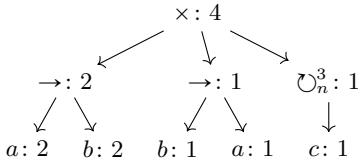
Applying Φ exhaustively with Φ_M requires executing this process a number of times. So long as the rule list R is solely reduction rules, then the size of the tree is monotonically decreasing. The worst case for time complexity is then



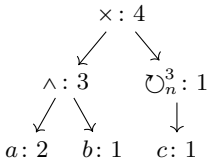
(a) Trace model $tm(L_E)$.



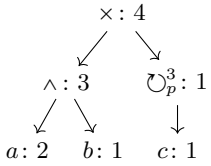
(b) Applying the *Choice similarity* rule FP.1.



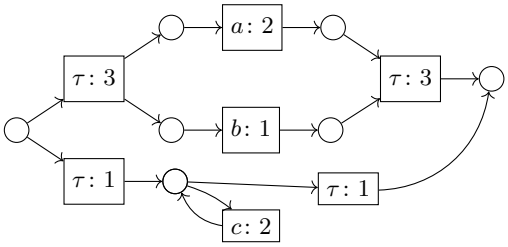
(c) *Loop roll* CO.2 (example *pe*).



(d) *Concurrent reduction* FPL.1.



(e) *Geometric abstraction* FPL.2



(f) *GSPN output* for $dtm(L_E)$.

Fig. 4: Discovery example using *dtm*.

also the best case for model size reduction and is bounded by the size of the trace model, $2||L||$. This yields $O(\Phi_M(L, R)) = O(||L||^3|R|)$. Translation to a GSPN with the *gspn* function is linear in the size of the tree (see Table 1). The overall worst-case time complexity is then dominated by the cubic term and is $O(||L||^3|R|)$.

4.3 Transformation Rules

Probabilistic Process Trees may be manipulated using transformation rules. In this section we organize and classify rules two ways: by information-preservation (using quality criteria), and by impact on determinism. After introducing some useful functions for merging and scaling PPTs, we then explain specific rules.

Classification By Quality Criteria. Transformation rules are classified using process model quality criteria of fitness, precision, and simplicity [1, p189], and by the criteria of stochastic information loss. Standard process model quality

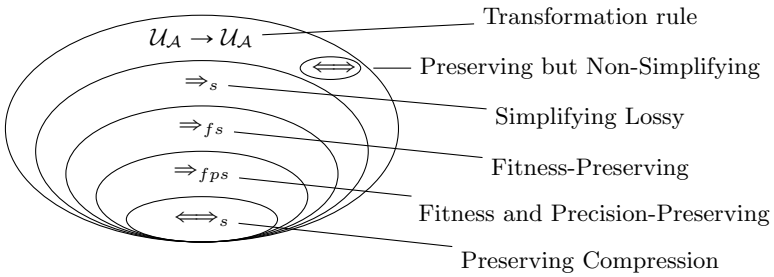


Fig. 5: Rule categories by information preservation.

criteria are control-flow criteria: they do not take the stochastic perspective into account.

Consider log L with language L_L and stochastic language Σ_L, and model M with language L_M and stochastic language Σ_M. Fitness is given by $ft(L_M, L_L) = \frac{|L_M \cap L_L|}{|L_L|}$. In defining precision, we have to account for infinitely many traces, due to the $\circlearrowleft_p$ construct [34]. Low-probability traces which are longer than the number of events in the log are filtered out in truncated language L_{TM}:

$$L_{TM} = \{t \in L_M \mid |t| < ||L|| \vee \Sigma_M(t) > \epsilon\} \text{ where } 0 < \epsilon \ll 1$$

Precision is then given by $pn(L_M, L_L) = \frac{|L_{TM} \cap L_L|}{|L_{TM}|}$.

As a categorization tool for process model transformation rules, fitness and precision are helpful in showing the loss or retention of information, even if they are insensitive to stochastic information. The classification of reduction rules is of particular interest, and necessary to maintain the monotonically simplifying property of the discovery algorithm in Section 4.2. We categorize reduction rules in four cuts: Preserving Compression, Fitness- and Precision-Preserving, Fitness-Preserving, and Simplifying Lossy, as seen in Figure 5. Another useful category of rules, Preserving but Non-Simplifying, change model structure without reducing the size of the model.

No Loss Of Fitness Or Precision Without Loss of Stochastic Information. There are no categories of "Fitness- and Stochastic Information-Preserving But Precision-Reducing" or "Precision- and Stochastic Information-Preserving but Fitness Reducing" transformation rules. Let stochastic fidelity between stochastic languages Σ_1, Σ_2 be $\forall t \in T \bullet \Sigma_1(t) = \Sigma_2(t)$. Models that no longer have stochastic fidelity have experienced stochastic information loss.

Theorem 1. *It is not possible to maintain fitness and reduce precision without also losing stochastic information from a model. Given log L, models M and M', and corresponding stochastic languages $\Sigma_M, \Sigma_{M'}$:*

$$ft(M, L) = ft(M', L) \wedge pn(M, L) > pn(M', L) \implies \exists t \bullet \Sigma_M(t) \neq \Sigma_{M'}(t)$$

Proof. Let Σ_L be the stochastic language for a trace model, which then has full stochastic information from the log. Let M' be a second model covering language $L_{M'}$ such that $ft(M, L) = ft(M', L) \wedge pn(M, L) > pn(M', L)$. By the fitness

definition, $\frac{|L_M \cap L_L|}{|L_L|} = \frac{|L_{M'} \cap L_L|}{|L_L|}$. As precision decreases, there must therefore be at least one new trace $t \in M' \wedge t \notin M$, which is equivalent to $\Sigma_M(t) = 0$ and $\Sigma_{M'}(t) > 0$. As probabilities must sum to one, some other trace $s \in M$ must have reduced in probability. $\exists s \in L_M \bullet \Sigma_{M'}(s) < \Sigma_M(s)$.
Case 1: Stochastic fidelity had been retained. $\Sigma_M(s) = \Sigma_L(s) \neq \Sigma_{M'}(s)$.
Case 2: Stochastic fidelity already lost. $\Sigma_M(s) \neq \Sigma_L(s)$. The trace t holds no information for restoring stochastic information on s, as t is not an element of the log L or covered by the original model M. □

If $\frac{|M'|}{|M|} < \frac{|L \cap M'|}{|L \cap M|}$, and a rule reduces fitness, then precision increases. This defines a sub-category of Simplifying Lossy Rules, however no useful concrete rules were found in this sub-category.

Classification By Determinism. PPTs are not constrained to describe deterministic languages. Non-determinism arises when the next symbol in a trace will satisfy multiple paths through a process tree. This can be shown trivially with the tree $\times(a\colon 1, a\colon 2)$. The trace model produced by function tm may also produce a non-deterministic tree, for example $tm([\langle a\rangle, \langle a, b\rangle]) = \times(a\colon 1, \rightarrow(a, b)\colon 1)\colon 2$. Determinism is a desirable property in an output model: it makes problems such as parsing and calculating the most probable path easier [38], and some important stochastic conformance techniques are constrained to deterministic models [19]. In this section we describe functions for calculating the determinism of a model, and how rules may preserve determinism.

As the operators $\rightarrow, \circlearrowleft_n$, and $\circlearrowleft_p$ are all sequential in form, the only PPT operators which may introduce non-determinism are $\times$ and $\wedge$. The function β reports whether a tree is deterministic.

$$\text{Let } \alpha\colon \mathcal{U}_\mathcal{A} \rightarrow \mathcal{P}(A) \text{ identify starting symbols}$$
$$\alpha(a\colon w) = \{a\} \text{ where } a \in A$$
$$\alpha(\oplus(U_1, ..., U_n)\colon w) = \begin{cases} \alpha(U_1) \text{ where } \oplus \in \{\rightarrow, \circlearrowleft_n, \circlearrowleft_p\} \\ \alpha(U_1) \cup ... \cup \alpha(U_n) \text{ where } \oplus = \times \\ \exp(U_1) \cup ... \cup \exp(U_n) \text{ where } \oplus = \wedge \end{cases}$$
$$\text{Let } \alpha_{st}\colon \mathcal{U}_\mathcal{A} \rightarrow \mathcal{P}(A) \text{ identify non-determinant sub-tree symbols}$$
$$\alpha_{st}(a\colon w) = \varnothing \text{ where } a \in A$$
$$\alpha_{st}(\oplus(U_1, ..., U_n)\colon w) = \begin{cases} \alpha(U_1) \cap ... \cap \alpha(U_n) \text{ where } \oplus \in \{\times, \wedge\} \\ \alpha(\oplus(U_1, ..., U_n)\colon w) \text{ otherwise} \end{cases}$$
$$\text{Let } \beta\colon \mathcal{U}_\mathcal{A} \rightarrow \mathbb{B} \text{ identify whether a tree is deterministic}$$
$$\beta(U) \text{ where } a \in A$$
$$\beta(\oplus(U_1, ..., U_n)\colon w) = \begin{cases} \forall i \bullet \beta(U_i) \text{ where } \oplus \in \{\rightarrow, \circlearrowleft_n, \circlearrowleft_p\} \\ \alpha_{st}(\oplus(U_1, ..., U_n)\colon w) = \varnothing \text{ where } \oplus \in \{\times, \wedge\} \end{cases}$$

$\beta(U)$ only has to visit each node at most once, so can complete in $O(|U|)$ time.

Table 2: PPT transformation rule classification by impact on determinism, given $U' = tr(U)$ for some rule tr.

Classification	Definition	Description
α-reducing	$\alpha(U') \subseteq \alpha(U) \wedge \alpha_{st}(U') \subseteq \alpha_{st}(U)$	Separates and restricts relevant symbols
Non-optional	No $\times$ or $\wedge$	Special case of α-stable
α-stable	$\alpha(U') = \alpha(U)$	No change to relevant symbols
β-trap	$\beta(U) \implies \beta(U')$	Never introduces non-determinism. Superset of preceding types.

As PPTs are a subset of Probabilistic Finite Automata (PFAs), Deterministic Probabilistic Process Trees (DPPTs) are Stochastic Deterministic Finite Automata (SDFAs) [38]. SDFAs are not closed under union [38] and DPPTs combinations are not closed under $\times$; trivially, $\times(a\colon 1, a\colon 2)$ combines two deterministic sub-trees. However DPPTs have the useful property of being closed under certain subsets of transformation rules.

Definition 8 (β-trap). *A β-trap is a transformation rule which preserves determinism: $\forall U \in \mathcal{U}_{\mathcal{A}}, tr\colon \langle \mathcal{U}_{\mathcal{A}} \to \mathcal{U}_{\mathcal{A}} \rangle \bullet tr$ is a β-trap $\iff \beta(U) \implies \beta(tr(U))$*

We use α and β to classify rules by their impact on determinism in Table 2.

Theorem 2. *DPPTs are closed under reduction by β-trap rules, so will not introduce non-determinism to a deterministic model.*

Proof. From the definition of β and β-trap, $\beta(U) \implies \beta(tr(U))$, so the composition of β-trap rules is itself a β-trap. □

DPPTs are closed under β-traprule composition, but not tree composition.

Theorem 3. *Application of α-stable or α-reducing rules to a sub-tree preserves the determinism of the parent tree, but a β-trap rule may not.*

Proof. Consider possible transformation rule $nb(\to(a\colon 1, a\colon 1, b\colon 1)) = \times(a\colon 1, b\colon 1)$. (Note this is not a rule used for our discovery algorithm.) The rule preserves determinism, as $\beta(nb(x)) \wedge \beta(x)$. However, $\alpha(\times(a\colon 1, b\colon 1)) = \{a, b\} \supset \alpha(\to(a\colon 1, a\colon 1, b\colon 1) = \{a\}$.

For α-stable and α-reducing rules tr, for process tree $\oplus(U_1, ..., U_n)$, as $\forall i \bullet \alpha(tr(U_i)) \subseteq \alpha(U_i)$, the intersection $\alpha(U_i) \cap \alpha(U_j)$ does not increase, so $\beta(U) \implies \beta(tr(U))$, the definition of a β-trap. □

These results do not guarantee the discovery of a deterministic model, but they do maintain the determinism of a reduced model once one is discovered.

Concrete Rules. The remainder of this section concerns concrete rules. We use $a, b \in A$ for activities and $u_1, u_2 \in U$ to represent PPTs or sub-trees. Weights are represented by $v_i, w_i \in \mathbb{R}^+$. Unweighted operators or activities are represented by x, y such that $x\colon w, y\colon v \in \mathcal{U}_{\mathcal{A}}$. In each subsection, we introduce the information-preservation category, then each rule in the category. The impact on determinism, using the system in Table 2, is also shown.

Helper Functions. These functions and relations help define transformation rules.

A scaling function, $\Gamma(U, \gamma)$ multiplies every weight in the tree by γ. Γ preserves α-stability. PPTs are *similar*, denoted $\cong_c$, when only weights need to be changed to make them strictly equal.

A stochastic merge function, Ψ, combines two similar trees by adding weights, $\circlearrowleft_p$ repetitions being scaled by relative weight. Ψ preserves α-stability, and also the control-flow fitness and precision of the input process trees, but loses stochastic information, unless $x = y$ for $\Psi(x, y)$.

Parameter Consolidation (Preserving but non-simplifying). The following associativity rules allow all PPTs to be transformed into equivalent trees with at most two children per operator. In our Toothpaste miner algorithms, such rules must only be used in combination with reduction rules, to maintain monotonicity of reduction; they act as meta-rules which allow other rules to be stated more concisely. We denote these rules as $\Longleftrightarrow$. They are information-preserving and perform no compression from left-to-right. They are all α-stable.

PC.1 $\rightarrow(s_1, s_2, ..., s_n) : w \Longleftrightarrow \rightarrow(s_1, \rightarrow(s_2, ..., s_n) : w) : w$
PC.2 $\times(s_1 : w_1, s_2 : w_2, ..., s_n : w_n) : v$
$\Longleftrightarrow \times(s_1 : w_1, \times(s_2 : w_2, ..., s_n : w_n) : v - w_1) : v$
PC.3 $\wedge(s_1 : w_1, s_2 : w_2, ..., s_n : w_n) : v$
$\Longleftrightarrow \wedge(s_1 : w_1, \wedge(s_2 : w_2, ..., s_n : w_n) : v - w_1) : v$
PC.4 The $\times$ and $\wedge$ operators are commutative. $\oplus(u, v) = \oplus(v, u)$
The remaining rules are accordingly stated using at most two operator parameters.

Preserving Compressions. The following rules are information-preserving reduction rules, achieving compression by using a smaller tree to describe the same stochastic language. They are denoted with $\Longleftrightarrow_s$ and are all non-optional deterministic.

CO.1 *Fixed loop identity.* $\circlearrowleft_n^1 (u) : w \Longleftrightarrow_s u : w$. This is used in reverse in FPL.7.
CO.2 *Fixed Loop roll.* $\rightarrow(x, x) : w \Longleftrightarrow_s \circlearrowleft_n^2 (x) : w$
$\rightarrow(x, \circlearrowleft_n^m (x)) : w \Longleftrightarrow_s \circlearrowleft_n^{m+1} (x) : w$
$\rightarrow(\circlearrowleft_n^m (x), x) : w \Longleftrightarrow_s \circlearrowleft_n^{m+1} (x) : w$
CO.3 *Silent sequence.* $\rightarrow(u, \tau) : w \Longleftrightarrow_s \rightarrow(\tau, u) : w \Longleftrightarrow_s u : w$
CO.4 *Silent concurrency.* $\wedge(u : w, \tau : v) : w + v \Longleftrightarrow_s \Gamma(u, \frac{w+v}{w})$
CO.5 *Fixed loop nesting.* $\circlearrowleft_n^n (\circlearrowleft_n^m (u)) : w \Longleftrightarrow_s \circlearrowleft_n^{nm} (u) : w$

Fitness and Precision-Preserving With Stochastic Information Loss. For these rules, stochastic information is preserved only where sub-trees are strictly equal, as for Ψ. They are denoted with $\Rightarrow_{fps}$. The determinism properties vary by rule.

FP.1 *Choice similarity reduction.* Merge choices between structurally similar trees. $\times(u_1, u_2) \Rightarrow_{fps} \Psi(u_1, u_2)$ where $u_1 \cong_c u_2$. This rule is α-stable.
FP.2 *Choice folding.* Pull up a common prefix into the head of a new sequence. $\times((\rightarrow(u_{x1}, u_2) : w_1), (\rightarrow(u_{x2}, u_3) : w_2) : w_1 + w_2$
$\Rightarrow_{fps} \rightarrow (\Psi(u_{x1}, u_{x2}), \times(u_2, u_3)) : w_1 + w_2$ where $u_{x1} \cong_c u_{x2}$. This rule is α-reducing. (Illustrated in Figure 6).

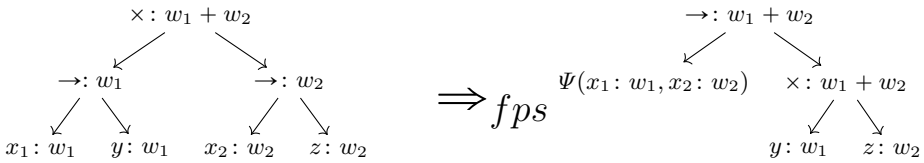


Fig. 6: *Choice folding* transformation rule for shared prefixes FP.2.

FP.3 *Choice folding suffixes* $\times((\rightarrow (u_1, u_{y1}) : w_1), (\rightarrow (u_2, u_{y2}) : w_2)) : w_1 + w_2$ $\Rightarrow_{fps} \rightarrow (\times(u_1, u_2), \Psi(u_{y1}, u_{y2})) : w_1 + w_2$ where $u_{y1} \cong_c u_{y2}$. This rule is α-stable.

FP.4 *Choice skip.* A common head is pulled into a sequence with a choice between the tail and a silent activity. $\times(x_1 : w_1, \rightarrow (x_2, y) : w_2) : v$ $\Rightarrow_{fps} \rightarrow (\Psi(x_1 : w_1, x_2 : w_2), \times(y : w_2, \tau : w_1)) : v$ where $x_1 : w_1 \cong_c x_2 : w_2$. This rule is α-reducing.

FP.5 *Choice skip suffix.* $\times(x_1 : w_1, \rightarrow (y, x_2) : w_2) : v$ $\Rightarrow_{fps} \rightarrow (\times(y : w_2, \tau : w_1), \Psi(x_1 : w_1, x_2 : w_2)) : v$ where $x_1 : w_1 \cong_c x_2 : w_2$. This rule is α-stable.

FP.6 *Concurrent similarity reduction.* Similar concurrent subtrees reduce to repetition. $(\wedge(x_1 : w, x_2 : v) : w + v)$ $\Rightarrow_{fps} \circlearrowleft_n^2 (\Psi(x_1 : w, x_2 : v)) : w + v)$ where $x_1 \cong_c x_2$. This rule is α-stable.

FP.7 *Concurrent subsumption.* Sequences already recognized as concurrent are pulled under that pattern. $\times(\rightarrow (x_1, y_1) : w_1, \wedge(x_2 : w_2, y_2 : w_3) : v)$ $\Rightarrow_{fps} \wedge(\Psi(x_1 : w_1, x_2 : w_2), \Gamma(\Psi(y_1 : w_1, y_2 : w_3), \frac{w_3}{w_1 + w_3})) : v$ where $x_1 \cong_c x_2 \wedge y_1 \cong_c y_2$. This rule is α-stable.

Fitness-Preserving Lossy Reductions. These rules preserve control-flow fitness of the input model with respect to a given log, but may reduce precision and stochastic information. They are denoted with $\Rightarrow_{fs}$.

FPL.1 *Concurrent reduction from choice sequences.* Concurrency is inferred when permutations of a given two-step sequence are seen. Generalizing concurrency involves re-scaling the weights of the merged sub-trees.
$\times(\rightarrow (u_{x1}, u_{y1}) : w, \rightarrow (u_{y2}, u_{x2}) : v) : w + v$
$\Rightarrow_{fs} \wedge(\Gamma(\Psi(u_{x1} : w, u_{x2} : v), \frac{w}{w+v}), \Gamma(\Psi(u_{y1} : w, u_{y2} : v), \frac{v}{w+v}) : w + v$
where $u_{x1} \cong_c u_{x2} \wedge u_{y1} \cong_c u_{y2}$.
When sub-trees are compound (that is, not activities), and applied across partial parameters, as in the binary statement here, there is concurrency generalization from a sample rather than complete evidence of concurrency. E.g., traces $\langle a, b, c\rangle, \langle b, a, c\rangle, \langle c, a, b\rangle$ are sufficient to reduce to $\wedge(a : 1, b : 1, c : 1)$. As all activities are already present in both children before the application of the rule, concurrent reduction is α-stable.

FPL.2 *Geometric Abstraction.* This rule combines fixed loops into a single probabilistic loop. Consider $\times(\circlearrowleft_n^{m_1} (u_1) : w_1, ..., \circlearrowleft_n^{m_n} (u_n) : w_n) : v$ where $\forall i, j \leqslant n \bullet u_i \cong_c u_j$. By definition of $\times$, $v = \Sigma_1^n w_i$. These loops are used as samples

in a geometric probability distribution.

$$\text{Probability of exit } P = \frac{\Sigma_1^n w_i}{\Sigma_1^n m_i w_i}$$

$$\text{Mean repetitions } \bar{\rho} = \frac{1}{P} = \frac{\Sigma_1^n m_i w_i}{\Sigma_1^n w_i}$$

Helper function μ averages n loops using a scaled fold with Ψ

$$\bar{u} = \mu(\circlearrowleft_n^{m_1}(u_1)\colon w_1, ..., \circlearrowleft_n^{m_n}(u_n)\colon w_n) = \circlearrowleft_p^{\bar{\rho}}(\Gamma(\Psi(\Gamma(u_1, m_1), \Psi(..., \Gamma(u_n, m_n))), P)\colon v$$

$$\text{Then } \times(\circlearrowleft_n^{m_1}(u)\colon w_1, ..., \circlearrowleft_n^{m_n}(u)\colon w_n)\colon v \Rightarrow_{fs} \mu(\circlearrowleft_n^{m_1}(u)\colon w_1, ..., \circlearrowleft_n^{m_n}(u)\colon w_n).$$

Geometric abstraction is non-optional deterministic.

FPL.3 *Choice Loop roll.* A tree is always a loop of length one, so can be incorporated in a loop by averaging. $\times(x_1\colon w_1, \circlearrowleft_p^{\rho}(x_2)\colon w_2) \Rightarrow_{fs} \circlearrowleft_p^{\bar{\rho}}(\mu(\circlearrowleft_p^1(x_1\colon w_1), \circlearrowleft_p^{\rho}(x_2)\colon w_2))\colon w_1 + w_2$ given $x_1 \cong_c x_2$ and where $\bar{\rho} = \frac{w_1 + \rho w_2}{w_1 + w_2}$. This rule is α-stable.

FPL.4 *Fixed Loop of Probability loops.* The sum of geometric distributions is a negative binomial distribution. This rule approximates with a geometric distribution of the same mean. $\circlearrowleft_n^m(\circlearrowleft_p^{\rho}(x))\colon w \Rightarrow_{fps} \circlearrowleft_p^{\rho(m-1)}(x)\colon w$ where $m > 1$.

The $m = 1$ case is handled by Fixed loop identity CO.1. This rule is in the non-optional determinism category.

FPL.5 *Probability Loop of Fixed loops.* $\circlearrowleft_p^{\rho}(\circlearrowleft_n^m(x))\colon w \Rightarrow_{fps} \circlearrowleft_p^{m\rho}(x)\colon w$. This rule is in the non-optional determinism category.

FPL.6 *Probabilistic Loop Nesting.* $\circlearrowleft_p^{\rho_1}(\circlearrowleft_p^{\rho_2}(x))\colon w \Rightarrow_{fps} \circlearrowleft_p^{\rho_1\rho_2}(x)\colon w$ by the product of expectations. This rule is in the non-optional determinism category.

FPL.7 *Loop Similarity Normalization.* Loops are not similar to their subtrees by $\cong_c$. However loops and their children can be usefully consolidated with some loss of information. Noting that $\circlearrowleft_n^1(u)\colon w \iff u\colon w$, we define loop similarity $\cong_L\colon \mathcal{U}_{\mathcal{A}} \leftrightarrow \mathcal{U}_{\mathcal{A}}$:

$$u_1 \cong_L \circlearrowleft_p^{\rho}(u_2) \Leftrightarrow u_1 \cong_c u_2$$

$$u_1 \cong_L \circlearrowleft_n^{m}(u_2) \Leftrightarrow u_1 \cong_c u_2$$

Reduction rules using $\cong_c$ each have a Fitness-Preserving Lossy ($\Rightarrow_{fs}$) variation using $\cong_L$ and replacement tree $\bar{u}$.

$$\text{For rule parameters } x_1\colon w_1 \cong_L \circlearrowleft_p^{\rho}(x_2\colon w_2)\colon w_2$$

$$\circlearrowleft_p^{\rho'}(\bar{u}) = \mu(\circlearrowleft_p^1(x_1\colon w_1), \circlearrowleft_p^{\rho}(x_2\colon w_2))$$

The consolidated tree $\bar{u}$ may replace u_1 and u_2 in a transformation rule tr where $u_1 \cong_L u_2$ and the resulting rule application still results in tree size reduction. Loop similarity is non-optional deterministic, but note the rule it is applied to may have a weaker determinism category, which will become the category of the final rule.

Simplifying Lossy Reductions. The last rule category abstracts or summarizes a PPT, from both a control flow and stochastic perspective, at the expense of control-flow fitness and precision. Such rules are useful for the management of noise and for allowing a user to tune the detail of the model for their specific use case. They are denoted with $\Rightarrow_s$, with one rule in this category.

SL.1 *Choice Pruning.* $\times(x\colon w_1, y\colon w_2)\colon v \Rightarrow_s x\colon v$ where $\frac{w_2}{v} < \epsilon$ for some supplied probability threshold ϵ. This rule is α-reducing.

5 Incremental Discovery and Optimisations

A simple Toothpaste Miner was introduced in Section 4.2, but can be limited when applied to streams, very large event logs, or noisy data. Incremental process model discovery is of interest for streams of events and very large event logs, e.g. in [18]. Better models may be achieved through other optimizations while maintaining tractability. Management of noise is another key process mining challenge [1, p185], which alternative rulesets can address within the overall Toothpaste Miner framework.

5.1 Incremental Discovery

The Φ_Δ algorithm adds a new trace to the existing model and applies reduction rules: $\Phi_\Delta\colon \mathcal{U}_\mathcal{A} \times \mathcal{U}_\mathcal{A} \times \langle \mathcal{U}_\mathcal{A} \to \mathcal{U}_\mathcal{A} \rangle \to \mathcal{U}_\mathcal{A}$.

$$\Phi_\Delta(x\colon w, tt\colon v, rs) = \Phi_M(\times(x\colon w, tt\colon v)\colon w + v, rs)$$

Definition 9 (Incremental Toothpaste miner). *Given trace $t \in \langle A \rangle$, rule sequence rs, existing model $U_M \in \mathcal{U}_\mathcal{A}$, and functions gspn and tm per Definition 7, incremental miner dinc: $\langle A \rangle \times \langle \mathcal{U}_\mathcal{A} \to \mathcal{U}_\mathcal{A} \rangle \times \mathcal{U}_\mathcal{A} \to (\mathcal{G} \times \mathcal{U}_\mathcal{A})$ is:*

$$dinc(t, rs, U_M) = (gspn(U_{M+1}), U_{M+1}),\ \textit{given } U_{M+1} = \Phi_\Delta(U_M, st(t), rs)$$
$$dinc(t, rs, \varnothing) = (gspn(st(t)), st(t))$$

An entire event log L may be presented as a stream to *dinc*, resulting in repeated invocations of Φ_Δ.

Definition 10 (Repeated Incremental Toothpaste Miner).

$$di\colon L \times \langle \mathcal{U}_\mathcal{A} \to \mathcal{U}_\mathcal{A} \rangle \to \mathcal{G}$$
$$(di(L, rs), pt) = dinc(t, rs, di(L - \{t\}, rs)),\ \textit{for some } t \in L$$
$$(di([\![t^1]\!], rs), pt) = dinc(t, rs, \varnothing)$$

5.2 Incremental Complexity

As for other Φ and *dtm* algorithms, the time complexity of *di* is dependent on the size of the tree. For the first trace t_1, the size of the initial model is $2|t_1| - 1$. In the worst case, the size of the process model increases over time, so that subsequent traces t_i add $2|t_i|$ nodes each. The time for each run of Φ_Δ is $2|t_i|^2|R|$. The overall worse case size for a log L is then

$$\begin{aligned} O_{mem}(di(L, rs)) &= \Sigma_{i=1}^{|L|} 2|t_i||R| = 2|R|\Sigma_{i=1}^{|L|}|t_i| \\ &= O(|R| \cdot ||L||), \text{ by the definition of} ||L|| \end{aligned}$$

An upper bound for time complexity can be found using Lemma 1.

Lemma 1. *For* $A \subseteq \mathbb{N}$, $\Sigma_{a\in A}a^2 \leqslant (\Sigma_{a\in A}a)^2$.

Proof. By induction over $|A|$; initial case $|A| = 1$, for which $a^2 = (a)^2$,

Show $\Sigma_{a\in A}a^2 \leqslant (\Sigma_{a\in A}a)^2 \implies \Sigma_{a\in A\cup\{b\}}a^2 \leqslant (\Sigma_{a\in A\cup\{b\}}a)^2$

$$(\Sigma_{a\in A\cup\{b\}}a)^2 = (\Sigma_{a\in A}a + b)^2 = (\Sigma_{a\in A}a)^2 + 2b(\Sigma_{a\in A}a) + b^2$$

$$\Sigma_{a\in A}a^2 \leqslant (\Sigma_{a\in A}a)^2 \implies \Sigma_{a\in A}a^2 + b^2 \leqslant (\Sigma_{a\in A}a)^2 + b^2 + 2b(\Sigma_{a\in A}a)\square$$

Then, $O_{time}(di(L, rs)) = \Sigma_{i=1}^{|L|}2|t_i|^2|R| = 2|R|\Sigma_{i=1}^{|L|}|t_i|^2 \leqslant 2|R|(\Sigma_{i=1}^{|L|}|t_i|)^2 \leqslant 2|R|||L||^2$, by definition of $||L||$. So $O_{time}(di(L, rs)) \leqslant O(||L||^2|R|)$.

Notably, the time complexity of the incremental algorithm *di* is quadratic, rather than the cubic complexity of reducing the entire trace model at once with *dtm*. Informally, the model is of a smaller size for more of the execution of the algorithm, with the time savings compounding. An important design trade-off remains, as stochastic information loss occurs with most reduction rules, and some classes of rules cause more information loss than others.

5.3 K Retries

Finding the minimal model with $\Phi(pt, rs)$ would require checking all $|rt|!$ permutations of reduction rules, so becomes intractable even for relatively small collections of rules. Rather than using the first full reduction, as in Definition 7, exploring an alternative K permutations, for small constant K, may yield a smaller model, without impacting computational complexity.

$$R_K = \{r \in \langle \mathcal{U}_\mathcal{A} \to \mathcal{U}_\mathcal{A}\rangle \mid \mathbf{ran}\ r = \mathbf{ran}\ rs\} \wedge |R_K| = K$$

$$\Phi_{MK}(pt, rs) = pt'$$

$$\text{where } |pt'| = \mathbf{min}(\{c \in \mathbb{N} \mid \exists p \in \mathcal{U}_\mathcal{A}, rs' \in R_K \wedge c = |\Phi(p, rs')|\})$$

5.4 Noise and Lossy Rules

As discussed in Section 4.3, reduction rules may be categorized according to the information loss they cause and their impact on process model quality criteria. This can impact incremental or full-trace reduction algorithms, and information losses may also compound with repeated rule applications. This is most severe for the incremental algorithm *dinc*, as summary models are local on the log presented so far, and do not benefit from the context of the full log.

As an example, the *Choice Pruning* rule SL.1 removes low probability activities. However, probabilities of activities will fluctuate when few traces have been processed, and this may remove nodes which are actually well-represented across a full log. Accordingly, discovery algorithm variant *dc* takes a cleanup ruleset as a separate parameter, and performs a penultimate cleanup phase, applying the cleanup rules only once.

Definition 11 (Toothpaste Miner with Cleanup (TMC)).

Given primary ruleset rs *and cleanup ruleset* cl,

$$dc\colon L \times \langle \mathcal{U}_\mathcal{A} \to \mathcal{U}_\mathcal{A}\rangle \times \langle \mathcal{U}_\mathcal{A} \to \mathcal{U}_\mathcal{A}\rangle \to \mathcal{G}$$

$dc(L, rs, cl) = \Phi_K(\Phi(di(L, rs), cl), rs)$ *is the TMC discovery algorithm*

The allocation of rules to main or cleanup rulesets is implementation-dependent.

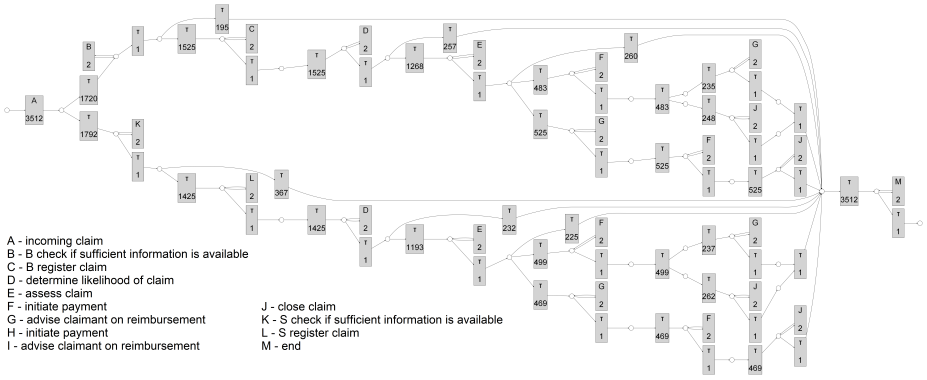


Fig. 7: *Teleclaims* process model.

6 Implementation and Evaluation

A prototype *Toothpaste Miner* was evaluated against existing stochastic process mining techniques using real-world logs. Results show good log conformance based on Earth-Movers distance [22], in feasible execution times, at the cost of more complex models.

6.1 Implementation

The prototype has been implemented in Haskell [1], extending other open source process mining tools [27]. Conversion from XES to a simpler, delimited text log format is done using Python and pm4py [6]. The implementation uses binary trees, to exploit the pattern-matching capabilities of Haskell. It maintains $\wedge$ and $\times$ nodes in lexical order for cheaper comparisons, and to limit traversal distance for similarity rules such as *Choice similarity* FP.1. Haskell allows for concise expression of transformation rules, as in Listing 1.1.

Listing 1.1: Choice similarity FP.1

```
choiceSim :: (Eq a) => PRule a
choiceSim (Node2 Choice x y n)
    | x =~= y = merge x y
choiceSim x = x
```

The extensions of incremental discovery and K retries are not included in the prototype. The noise-reducing choice pruning rule SL.1 is not included, and loop similarity FPL.7 is only partially applied. All fixed loops $\circlearrowleft_n$ are converted to probabilistic loops $\circlearrowleft_p$. Rules in Section 4.3 are otherwise included.

6.2 Evaluation Design

In order to evaluate the potential practical use of our technique, we compare it to established stochastic discovery techniques in the literature. K-fold cross validation ($k = 5$) was used on three logs, and results compared using stochastic quality criteria, simplicity and computation time, summarized in Table 3.

We evaluated all stochastic process discovery techniques where, to our knowledge, a public implementation was available. GDT_SPN discovery [30], weight

[1] Source code is accessible at https://github.com/adamburkegh/toothpaste

Table 3: Evaluation Design

Logs	Techniques	Measures	Environment
BPIC 2013 closed	Toothpaste (this paper)	tEMSC 0.8 [22]	Windows 10
Sepsis	GDT_SPN Discovery [30]	Entity count	2.3GHz CPU
Teleclaims [1]	walign-inductive [9]	Computation time	50 Gb memory
	walign-split [9]		GHC 8.8.4
	wfreq-inductive [9]		JDK 1.8.0_222
	wfreq-split [9]		Python 3.8.3
	wpairscale-split [9]		
	wpairscale-inductive [9]		
	Trace model		

estimator [9] and Inductive Miner [20] implementations are from ProM 6.9 (development branch Jan 2021). Stochastic weight estimator combinations were chosen where previous results [9] had shown meaningful differences. Logs were selected to represent multiple domains. All logs are publicly available [2]. BPIC2013 and Sepsis are real-life logs, and Teleclaims is an established dataset [1, p243].

Stochastic quality measures used were Earth Movers' Distance (tEMSC) [22] with 0.8 probability mass. Other stochastic measures such as [19] were restricted to deterministic nets. Entity count is used to measure complexity.

6.3 Results and Discussion

A selection of evaluation results[3] are shown in Table 4. The output from the full Teleclaims log is shown in Figure 7. For some k-fold logs and models, Earth Movers' Distance errored due to memory limits, or the calculation was timed out after 5 hours. No value reflects no result from any k-fold log; otherwise, partial results from the remaining logs have been used.

The Toothpaste Miner prototype shows a trade-off of improved Earth Movers' Distance against longer running times and higher model complexity. This is analogous to the trade-off of fitness and precision against complexity and run-time often seen with region-based control-flow miners. For reference log Teleclaims, a human-readable process model was discovered in four seconds.

Some rules did not fire during the evaluations, which suggests the value of optimizations in Section 5.4, where certain rules are only applied during later phases of discovery. Real-life logs show sensitivity to the ordering of choice rules versus log formation rules, with marked model differences depending on rule sequence. This may reflect the partial implementation of loop similarity FPL.7 in the prototype. For Teleclaims (see Figure 7), some redundancy is apparent due to prototype limitations in similarity identification for larger sub-trees.

On more challenging real-life logs, the prototype returned within two minutes, a shorter time than the existing GDT_SPN technique. For the BPIC2013 log, the additional complexity did not achieve a marked quality improvement. For the Sepsis log, it was able to retain a very similar $tEMSC$ to the trace model

[2] BPIC2013 and sepsis logs: https://data.4tu.nl/. Teleclaims: http://www.processmining.org/event_logs_and_models_used_in_book

[3] Full results are available at https://github.com/adamburkegh/toothpaste

Miner	Log	Duration (ms)	Entities	tEMSC 0.8
GDT_SPN discovery	BPIC2013 closed	203	25	0.387
Toothpaste	BPIC2013 closed	3071	315	0.668
Trace	BPIC2013 closed	31	2417	0.9334
walign-inductive	BPIC2013 closed	194	6	0.6756
walign-split	BPIC2013 closed	56	14	0
wfreq-inductive	BPIC2013 closed	160	6	0.7346
wfreq-split	BPIC2013 closed	18	14	0.6188
wpairscale-inductive	BPIC2013 closed	18	6	0.7364
wpairscale-split	BPIC2013 closed	15	14	0.8753
GDT_SPN discovery	Sepsis	10147434	95	-
Toothpaste	Sepsis	84551	2992	0.7019
Trace	Sepsis	25	5877	0.7029
walign-inductive	Sepsis	497	40	-0
walign-split	Sepsis	478	51	0.3645
wfreq-inductive	Sepsis	178	40	-
wfreq-split	Sepsis	25	51	0.5108
wpairscale-inductive	Sepsis	34	40	0.6664
wpairscale-split	Sepsis	37	51	-
GDT_SPN discovery	Teleclaims	453	60	-
Toothpaste	Teleclaims	3564	121	-
Trace	Teleclaims	61	17754	0.9771
walign-inductive	Teleclaims	238	28	0.3529
walign-split	Teleclaims	137	56	0.0931
wfreq-inductive	Teleclaims	203	28	0.5105
wfreq-split	Teleclaims	59	56	0.6301
wpairscale-inductive	Teleclaims	62	28	0.5143
wpairscale-split	Teleclaims	68	56	0.6299

Table 4: Evaluation results.

with a smaller entity footprint, though the final model is not human-readable. The reductions used may form part of other discovery or conformance strategies, say as a post-processing step.

7 Conclusion

Stochastic Petri Nets are powerful modelling tools with wide applicability. Automatically discovered stochastic process models, in turn, can help understand and improve organizations. In this paper we presented the *Toothpaste Miner* framework for discovering and reasoning about stochastic process models in the context of process mining. We shared both batch and incremental discovery algorithms, showed they were computationally tractable, and would maintain determinism in models once such a model was discovered. A classification scheme relating transformation rules and process mining quality measures was articulated. Lastly we discussed an implementation of the discovery technique, with an empirical evaluation showing close to trace-model levels of similarity to real-life logs, with significantly less required model entities.

Future work in this area may investigate algorithms guaranteed to output deterministic stochastic models, simpler process models with better human-readability, and solutions where the choice of ruleset allows for constraining solutions by particular quality parameters. Other extensions could support additional statistical distributions and timed transitions.

References

1. van der Aalst, W.: Process Mining: Data Science in Action. Springer-Verlag, Berlin Heidelberg, 2 edn. (2016)
2. Anastasiou, N., Horng, T.C., Knottenbelt, W.: Deriving generalised stochastic Petri net performance models from high-precision location tracking data. In: Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools. pp. 91–100. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2011)
3. Baum, L.E., Petrie, T., Soules, G., Weiss, N.: A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. The annals of mathematical statistics **41**(1), 164–171 (1970), publisher: JSTOR
4. Bause, F., Kritzinger, P.: Stochastic Petri Nets: An Introduction to the Theory. Vieweg+Teubner Verlag (2002)
5. Bellodi, E., Riguzzi, F., Lamma, E.: Statistical relational learning for workflow mining. Intelligent Data Analysis **20**(3), 515–541 (Jan 2016), publisher: IOS Press
6. Berti, A., van Zelst, S.J., van der Aalst, W.M.P.: Process Mining for Python (PM4Py) : Bridging the Gap Between Process- and Data Science. In: ICPMD 2019, ICPM Demo Track 2019 : proceedings of the ICPM Demo Track 2019, co-located with 1st International Conference on Process Mining (ICPM 2019) : Aachen, Germany, June 24-26, 2019 / edited by Andrea Burattin (Technical University of Denmark, Kgs. Lyngby, Denmark), Artem Polyvyanyy (The University of Melbourne, Melbourne, Australia), Sebastiaan van Zelst (Fraunhofer Institute for Applied Information Technology (FIT), Sankt Augustin, Germany). CEUR workshop proceedings, vol. 2374, pp. 13–16. RWTH Aachen, Aachen, Germany (Jun 2019), backup Publisher: 1st International Conference on Process Mining, Aachen (Germany), 24 Jun 2019 - 24 Jun 2019
7. Bezem, M., Klop, J., Barendsen, E., de Vrijer, R., Terese: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (2003)
8. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible Predictive Models for Business Processes. MIS Q. **40**(4), 1009–1034 (2016)
9. Burke, A., Leemans, S.J.J., Wynn, M.T.: Stochastic Process Discovery By Weight Estimation. In: 2020 International Conference on Process Mining (ICPM) (2020), press
10. Carmona, J., Cortadella, J., Kishinevsky, M.: New Region-Based Algorithms for Deriving Bounded Petri Nets. IEEE Transactions on Computers **59**(3), 371–384 (Mar 2010), conference Name: IEEE Transactions on Computers
11. Carrasco, R.C.: Accurate computation of the relative entropy between stochastic regular grammars. RAIRO-Theoretical Informatics and Applications **31**(5), 437–444 (1997), publisher: EDP Sciences
12. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: Carrasco, R.C., Oncina, J. (eds.) Grammatical Inference and Applications. pp. 139–152. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (1994)
13. Fünfgeld, S., Holzäpfel, M., Frey, M., Gauterin, F.: Stochastic Forecasting of Vehicle Dynamics Using Sequential Monte Carlo Simulation. IEEE Transactions on Intelligent Vehicles **2**(2), 111–122 (Jun 2017), conference Name: IEEE Transactions on Intelligent Vehicles
14. Geman, S., Geman, D.: Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. IEEE Transactions on Pattern Analysis and Ma-

chine Intelligence **PAMI-6**(6), 721–741 (Nov 1984), conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence
15. Hu, H., Xie, J., Hu, H.: A novel approach for mining stochastic process model from workflow logs. Journal of Computational Information Systems **7**(9), 3113–3126 (2011)
16. Janssenswillen, G., Depaire, B., Faes, C.: Enhancing Discovered Process Models using Bayesian Inference and MCMC. In: Proceedings of the 2020 BPI Workshop (2020)
17. Leclercq, E., Lefebvre, D., El Medhi, S.O.: Identification of timed stochastic petri net models with normal distributions of firing periods. IFAC Proceedings Volumes **42**(4), 948–953 (2009), publisher: Elsevier
18. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable Process Discovery with Guarantees. In: Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., Ma, Q. (eds.) Enterprise, Business-Process and Information Systems Modeling. pp. 85–101. Lecture Notes in Business Information Processing, Springer International Publishing, Cham (2015)
19. Leemans, S.J.J., Polyvyanyy, A.: Stochastic-Aware Conformance Checking: An Entropy-Based Approach. In: Dustdar, S., Yu, E., Salinesi, C., Rieu, D., Pant, V. (eds.) Advanced Information Systems Engineering: 32nd International Conference, CAiSE 2020, Proceedings. pp. 217–233. Springer, Switzerland (Jan 2020), conference Name: International Conference on Advanced Information Systems Engineering Meeting Name: International Conference on Advanced Information Systems Engineering
20. Leemans, S.J., Fahland, D., van der Aalst, W.M.: Discovering block-structured process models from event logs-a constructive approach. In: International conference on applications and theory of Petri nets and concurrency. pp. 311–329. Springer (2013)
21. Leemans, S.J., Poppe, E., Wynn, M.T.: Directly Follows-Based Process Mining: Exploration & a Case Study. In: 2019 International Conference on Process Mining (ICPM). pp. 25–32 (Jun 2019)
22. Leemans, S.J., Syring, A.F., van der Aalst, W.M.: Earth movers' stochastic conformance checking. In: International Conference on Business Process Management. pp. 127–143. Springer (2019)
23. Liesaputra, V., Yongchareon, S., Chaisiri, S.: Efficient Process Model Discovery Using Maximal Pattern Mining. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) Business Process Management. pp. 441–456. Lecture Notes in Computer Science, Springer International Publishing, Cham (2015)
24. Maggi, F.M., Montali, M., Peñaloza, R.: Probabilistic Conformance Checking Based on Declarative Process Models. In: Advanced Information Systems Engineering. pp. 86–99. Springer, Cham (Jun 2020)
25. Marsan, M.A., Balbo, G., Bobbio, A., Chiola, G., Conte, G., Cumani, A.: The effect of execution policies on the semantics and analysis of stochastic Petri nets. IEEE Transactions on Software Engineering **15**(7), 832–846 (Jul 1989), conference Name: IEEE Transactions on Software Engineering
26. Matsuno, H., Doi, A., Nagasaki, M., Miyano, S.: Hybrid petri net representation of gene regulatory network. In: Biocomputing 2000, pp. 341–352. World Scientific (Dec 1999)
27. Mokhov, A., Carmona, J., Beaumont, J.: Mining Conditional Partial Order Graphs from Event Logs. In: Koutny, M., Desel, J., Kleijn, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency XI, pp. 114–136. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2016)

28. Moreira, C., Haven, E., Sozzo, S., Wichert, A.: Process mining with real world financial loan applications: Improving inference on incomplete event logs. PLOS ONE **13**(12), e0207806 (Dec 2018), publisher: Public Library of Science
29. Polyvyanyy, A., Moffat, A., García-Bañuelos, L.: An Entropic Relevance Measure for Stochastic Conformance Checking in Process Mining. arXiv e-prints **2007**, arXiv:2007.09310 (Jul 2020)
30. Rogge-Solti, A., van der Aalst, W.M.P., Weske, M.: Discovering Stochastic Petri Nets with Arbitrary Delay Distributions from Event Logs. In: Lohmann, N., Song, M., Wohed, P. (eds.) Business Process Management Workshops. pp. 15–27. Lecture Notes in Business Information Processing, Springer International Publishing, Cham (2014)
31. Rogge-Solti, A., Weske, M.: Prediction of business process durations using non-Markovian stochastic Petri nets. Information Systems **54**, 1–14 (Dec 2015)
32. Secretary, I.C.: Information technology - Z formal specification notation - Syntax, type system and semantics. Standard, International Organization for Standardization, Geneva, CH (Mar 2002), volume: 2002
33. Silva, R., Zhang, J., Shanahan, J.G.: Probabilistic workflow mining. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. pp. 275–284. KDD '05, Association for Computing Machinery, Chicago, Illinois, USA (Aug 2005)
34. Tax, N., Lu, X., Sidorova, N., Fahland, D., van der Aalst, W.M.P.: The imprecisions of precision measures in process mining. Inf. Process. Lett. **135**, 1–8 (2018). https://doi.org/10.1016/j.ipl.2018.01.013, https://doi.org/10.1016/j.ipl.2018.01.013
35. Thierry-Mieg, Y.: Structural Reductions Revisited. In: Janicki, R., Sidorova, N., Chatain, T. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 303–323. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020)
36. Thollard, F., Dupont, P., De La Higuera, C.: Probabilistic DFA inference using Kullback-Leibler divergence and minimality. pp. 975–982 (Jun 2000)
37. Verwer, S., Eyraud, R., De La Higuera, C.: PAutomaC: a probabilistic automata and hidden Markov models learning competition. Machine learning **96**(1-2), 129–154 (2014), publisher: Springer
38. Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., Carrasco, R.: Probabilistic finite-state machines - part II. IEEE Transactions on Pattern Analysis and Machine Intelligence **27**(7), 1026–1039 (Jul 2005), conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence
39. Vidal, E., Thollard, F., Higuera, C.d.l., Casacuberta, F., Carrasco, R.C.: Probabilistic finite-state machines - part I. IEEE Transactions on Pattern Analysis and Machine Intelligence **27**(7), 1013–1025 (Jul 2005), conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence
40. Wang, X., Chen, G., Zhao, Q., Guo, Z.: Reduction of Stochastic Petri Nets for Reliability Analysis. In: 2007 8th International Conference on Electronic Measurement and Instruments. pp. 1–222–1–226 (Aug 2007), iSSN: null
41. Weisberg, M.: Simulation and similarity : using models to understand the world. Oxford Univ Press (Feb 2013)
42. Zhou, M., Venkatesh, K.: Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach. World Scientific (1999)