Enjoy the Silence, Part II: Probability-Based Queries on Stochastic Labelled Petri Nets

Sander J.J. Leemans[1,2], Marco Montali[3], Timo Gersing[1], Felix Engelhardt[1], and Natalia Sidorova[4]

[1] RWTH Aachen University, Germany
[2] Fraunhofer, Germany
[3] Free University of Bozen-Bolzano, Italy
[4] Eindhoven University of Technology

Abstract. A stochastic process model combines control flow and stochasticity in a single representation. Answering queries on the behaviour and probabilities of such a model is essential not only for analysis and verification, but also towards stochastic process mining, in which the frequency of events and traces is explicitly taken into account. In this paper, we focus on probability-based queries on stochastic process models, dealing with questions like "what are the 10 most likely traces?" and "what are the traces with a probability higher than 1%?", and "what are the most likely traces that together cover 80% of the behaviour in the model?". We formalise these queries in the setting where the model is represented as a stochastic labelled Petri net with repeated labels and silent transitions. We provide a representation of the stochastic deterministic state space induced by the net suitable for answering probability-based queries, and introduce an algorithm to do so. Finally, we implement our approach and evaluate its applicability and feasibility on real-life event logs.

Keywords: stochastic labelled Petri net · stochastic process mining · probability queries

1 Introduction

Capturing dynamic systems by combining behaviour and stochasticity has a long tradition in the theory and applications of Petri nets [4,24]. Interest in these models has recently resurged within process science, in order to capture business processes using stochasticity to quantitatively tackle uncertainty on the duration of activities and/or the way deferred choices are resolved.

In modelling and mining, dealing with stochastic processes is central. Process mining aims to provide business analysts with techniques to analyse business processes, based on traces (sequences of process steps) of process instances recorded in information systems. A process model, such as a Petri net, describes the process control-flow and can be obtained from an event log through automated process discovery, from domain knowledge or from industry process benchmarks [1,8]. While trace frequencies have always been central when analysing

processes, e.g., to distinguish and compare common and infrequent behaviour and to single out outliers, traditional process mining techniques do not convey this information at the model level. In particular, a decision point in the process is described as a non-deterministic choice in the Petri net, while trace frequencies in the log may be used to provide more refined, quantitative information.

This missing link has recently been tackled in *stochastic process mining*, where non-deterministic choices in process models are refined into stochastic choices that quantify the relative probability of selecting a route from the available options, moment by moment. Consequently, different challenges emerge with frequencies, stochastic choices and other forms of uncertainty [5].

When dealing with stochastic process modelling and mining, stochastic Petri nets constitute a natural candidate formalism, with three essential features that are not readily supported by traditional approaches [4,24]:

- Transitions are optionally labelled with activity names, and different transitions may have the same labels.
- Unlabelled (*silent*) transitions represent internal steps in the process, which are not logged. Silent transitions are omnipresent in process mining [1] to represent optional behaviour and process gateways (i.e. routing points).
- Process executions consist of unbounded, yet finitely many, steps, hence nets need to have a clear notion of termination, with a probability of termination attached. We solve this by treating every deadlock as a final marking.

These three features are tackled in so-called *stochastic labelled Petri nets* (SLPNs) [19], which have been extensively targeted in process mining to tackle discovery [27,3,14,18], conformance checking [15,20,19] and performance analysis [14]. However, models with unbounded state spaces and livelocks, i.e., progression is made but a final state cannot be reached, are typically not supported.

Conducting model analysis and verification in this spectrum, as well as dealing with process mining tasks, calls for analytic techniques of *querying* SLPNs and retrieving combined information on probabilities and model behaviour. Additionally, queries can be used to inspect a process model and extract useful insights, and constitute an essential building block to compute conformance and performance indicators. There are three main classes of queries: *state queries* - returning the probability that the process reaches a given marking; *behavioural queries* - returning the probability that the process generates a given trace or a trace that satisfies a given temporal specification; and *probability queries* - returning traces that meet a given condition on probabilities.

In particular, [19] has shown that state and behavioural queries can be homogeneously answered over *bounded* SLPNs by combining techniques from Markov chain analysis and formal verification. These queries are in turn useful to compute several conformance metrics that can otherwise only be approximated [22,23]. Such queries are particularly difficult to answer due to the presence of silent transitions, duplicated activities, and livelocks. The queries supported in [19] all take as input a given state or behaviour and return a probability mass. This leaves an important question, which we tackle in this paper: *how to answer probability-based queries*, that is, queries whose input refers to a probabil-

ity, and whose output is a set of most-likely behaviour that fills that probability mass. Importantly, these ideas are *independent of the adopted stochastic modelling formalism* (e.g., SLPNs), but touch the core of the modelled stochastic behaviour. Examples of such queries are: (1) What are the 10 most-likely traces in the SLPN? (2) What are the traces of the SLPN that have a probability higher than 1%? (3) What are the most-likely traces that together cover 80% of the model? (4) After executing a trace prefix, in what markings can the system be with what probability? All these queries can be generalised to one problem:
Problem: answering probability queries. Given a stochastic process model and a stopping criterion, find all traces in decreasing order of probability such that the criterion is met.

Probability queries enable process mining tasks related to: (1) *model inspection*, to obtain information on models that are too complex to be analysed manually; (2) *model validation* based on logs, such as sanity checks on the generalisation of stochastic process discovery techniques, for example to check whether the most frequent traces in the log remain so in the model; (3) *prediction*, querying the most likely outcomes given a prefix of a trace; and (4) *conformance checking*, since trace probabilities can be computed linearly on top of the structures we build to answer probability-based queries.

In this paper, we provide algorithmic techniques to answer probability queries over SLPNs, in a more general setting than that of [19] as we deal with *unbounded* nets with livelocks (Section 3). Reasoning directly over traces requires to on-the-fly removal of silent moves and merge distinct moves with identical labels, which is not supported by previous techniques. We provide a representation of the stochastic state space induced by the SLPNs that natively addresses these two aspects (Section 4). We then introduce an algorithm that iteratively explores such a state space to answer different types of probability queries (Section 5), guaranteeing termination in the large class of SLPNs without unbounded silent cycles under a conjecture – that is, loops of silent transitions that lead to unbounded markings (cf. Definition 10). The algorithm comes with an open-source implementation, which we use to evaluate applicability and feasibility of our approach on real-life event logs (Section 6). Finally, we relate our approach to existing literature (Section 7) and conclude the paper (Section 8).

2 Preliminaries

A weighted set is a function $M : E \to \mathbb{R}_{0^+}$ from a set E of elements to the non-negative real numbers, mapping each element in E to its weight in the set. We denote with $\boldsymbol{M} \in \mathbb{R}_{0^+}^{|E|}$ the incidence vector of a weighted set M. For weighted sets M and M' over E, we define multiset union $(M \uplus M')(x) = M(x) + M'(x)$ and difference $(M \setminus M')(x) = \max(0, M(x) - M'(x))$ for $x \in E$. We say that M is a subset of M', written $M \Subset M'$, if $\forall_x M(x) \leq M'(x)$. In case all weights are non-negative integers, we refer to the weighted set as a *multiset*. The set of all multisets over E is denoted by $E^{\mathbb{N}}$.

A probability distribution (*q-set*) $Q\colon E \to \mathbb{R}_{0+}$ is a weighted set whose elements' weights sum to one: $\sum_{e \in E} Q(e) = 1$. We denote with $\widetilde{Q}$ the set $\{e \mid Q(e) > 0\}$ of the elements with a positive probability in Q.

Definition 1 (Stochastic language). *A* trace *is a sequence of* activities. *A* stochastic language *is a weighted set of traces, such that the sum of their probabilities is less than or equal to 1. A stochastic language is* complete *if it is a q-set, i.e., a probability distribution.*

A *Markov chain* is a directed graph whose edges are annotated with probabilities, that is, a tuple (S, p) where S is a possibly infinite set of states and $p\colon S \times S \to [0, 1]$ a function such that for every $s \in S$, $\sum_{s' \in S} p((s, s')) = 1$.

In this paper, we employ *absorbing* Markov chains, that is, Markov chains where states are partitioned into two sets: a set of *absorbing states* containing states without any successor (or, equivalently, having only a self-loop), and a set of *transient states* containing non-absorbing states that are directly or indirectly connected to at least one absorbing state.

3 Stochastic Labelled Petri Nets

We define stochastic labelled Petri nets (SLPNs) and their language. Furthermore, we explore several challenging aspects of SLPNs, which are not typically considered in process mining contexts [27,19]: non-positive transition weights (to accomodate dynamic weights in future work), livelocks and unboundedness.

Definition 2 (Stochastic labelled Petri net). *A* stochastic labelled Petri net *(SLPN) is a tuple $(P, T, F, M_0, \Sigma, \lambda, w)$ in which P is a finite set of places, T is a finite set of transitions such that $P \cap T = \emptyset$, and $F \in ((P \times T) \cup (T \times P))^{\mathbb{N}}$ is a flow relation. A* marking $M \in P^{\mathbb{N}}$ *is a multiset of places (indicating* tokens *on places), and we call $M_0 \in P^{\mathbb{N}}$ the initial marking. Σ is a finite alphabet of activities and $\lambda\colon T \to \Sigma \cup \{\tau\}$ is a labelling function mapping the transitions to the activities or to the silent transition $\tau \notin \Sigma$. Furthermore, $w\colon T \to \mathbb{R}$ is a weight function on the transitions.*

Figure 1 shows an example of an SLPN.

We write ${}^\bullet t = [p \mid (p, t) \in F] \in P^{\mathbb{N}}$ and $t^\bullet = [p \mid (t, p) \in F] \in P^{\mathbb{N}}$ for the *preset* and *postset* of a transition $t \in T$. A transition $t \in T$ is *enabled* for a marking M if ${}^\bullet t \trianglelefteq M \wedge w(t) > 0$. For an SLPN in a marking M, we write $M \xrightarrow{a} M'$ if any transition t with $\lambda(t) = a$ is enabled in M such that firing t leads to M', that is $M' = M \uplus t^\bullet \setminus {}^\bullet t$. Similarly, $M \xrightarrow{\tau} M'$ indicates that a silent transition can fire in M and lead to M'.

The probability that a transition t fires in a marking M is inversely proportional to the summed weight of the enabled transitions:

Definition 3 (Likelihood of transition firing). *Let $(P, T, F, M_0, \Sigma, \lambda, w)$ be an SLPN, let $M \in P^{\mathbb{N}}$ be a marking, and let $t \in T$ be a transition that is enabled in M. Then, the probability that t fires in M is $p(M, t) = \frac{w(t)}{\sum_{t' \in T : {}^\bullet t' \trianglelefteq M \wedge w(t') > 0} w(t')}$.*

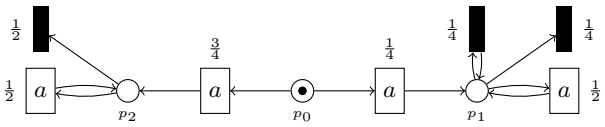


Fig. 1: An irregular SLPN. Silent transitions are black.

As usual, we call a marking M in which no transition is enabled a *deadlock*:

Definition 4 (Deadlock). *Let $(P, T, F, M_0, \Sigma, \lambda, w)$ be an SLPN, and let M be a marking. Then, M is a deadlock if and only if $\nexists_{t \in T} {}^\bullet t \trianglelefteq M \wedge w(t) > 0$.*

A *partial path* $\langle t_1 \dots t_n \rangle$ of an SLPN is formed by transitions that sequentially bring the SLPN from its initial marking M_0 to the markings $M_1, \dots, M_n$ with $\forall_{1 \leq i \leq n} M_i = M_{i-1} \uplus t_i^\bullet \setminus {}^\bullet t_i$ such that each transition t_i is enabled by M_{i-1}. The probability of a path is the product of the probabilities of transitions it consists of. A *complete path* is a partial path that ends in a deadlock. A (partial) *trace* is a (partial/complete) path projected by λ to labels of Σ, dropping silent transitions and retaining only its visible steps. The q-set of traces of an SLPN, denoted with $\mathcal{L}$, is its stochastic language. We say that a marking M_j is *reachable* from a marking M_i if there exists a partial path from M_i to M_j.

3.1 SLPNs with livelocks

Let M be a reachable marking from the initial marking M_0 of an SLPN; if it is impossible to reach a deadlock marking from M, then M is part of a *livelock*. An SLPN with a livelock has an incomplete stochastic language, witnessing that with some probability the modelled system keeps moving and can never complete.

Definition 5 (Livelock). *Let $(P, T, F, M_0, \Sigma, \lambda, w)$ be an SLPN, and let M be a marking that is reachable from M_0. M is part of a* livelock *($\bigcirc(M)$) if and only if there is no marking M' that is a deadlock and that is reachable from M.*

The labelling of transitions and the stochastic perspective are not relevant for livelocks. Therefore, decidability results of the livelock-freedom of standard Petri nets carry over to SLPNs directly. In particular, livelock-freedom is decidable [9, thms 3.2.25 & 3.2.26], though practical decision algorithms are expensive.

A particular type of livelock is the silent livelock, in which the modelled process can only progress internally, that is, through silent transitions, but cannot fire a visible transition anymore, and cannot terminate:

Definition 6 (Silent livelock). *Let $(P, T, F, M_0, \Sigma, \lambda, w)$ be an SLPN, and let M be a marking that is reachable from M_0 and that is part of a livelock. Let $\mathcal{M}$ be the set of all markings reachable from M. If for all $M' \in \mathcal{M}$ it holds that only silent transitions are enabled in M', then M is part of a silent livelock.*

While it is decidable whether a marking is part of a livelock, this is not a cheap computation [9]. Therefore, we consider a special case of livelock that is

Algorithm 1 Non-decreasing livelock decision

1: **procedure** NON-DECREASING LIVELOCK(SLPN $(P, T, F, M_0, \Sigma, \lambda, w)$, marking M)
2: $\mu \leftarrow \langle M \rangle$ ▷ sequence of markings
3: **while** $t \leftarrow$ only transition enabled in M **do**
4: $M \leftarrow M \uplus t^\bullet \setminus {}^\bullet t$ ▷ marking after firing t
5: $\mu \leftarrow \mu \cdot \langle M \rangle$
6: **if** $\mu = \langle \dots, M_k, \dots, M_\ell \rangle$ with $M_k \trianglelefteq M_\ell$ **then** ▷ we are in a cycle
7: $\Delta \leftarrow M_\ell \setminus M_k$ ▷ difference by cycle
8: **return** $\forall_{k \le i < \ell} \{t' \in T \mid w(t) > 0 \wedge {}^\bullet t' \trianglelefteq M_i \uplus \biguplus_{j=1}^{\infty} \Delta\} = 1$
9: ▷ cycling enables no other transition
10: **end if**
11: **end while**
12: **return** false
13: **end procedure**

cheaper to compute: in a *non-decreasing* livelock, at any point in the livelock, only one transition is enabled, and thus the probability of firing that transition is 1. As such, in a non-decreasing livelock, the probablity never decreases:

Definition 7 (Non-decreasing livelock). *Let $\langle t_1, \dots t_n \rangle$ be a partial path with $\langle M_0, \dots M_n \rangle$ its markings, such that M_n is part of a livelock. If there is only one infinite partial path of which $\langle t_1, \dots t_n \rangle$ is a prefix, then M_n is part of a* non-decreasing *livelock.*

An efficient algorithm to decide whether marking M is part of a non-decreasing livelock is shown in Algorithm 1. Intuitively, the algorithm walks along the cycle, and at every step only one transition can be enabled (line 3). At some point (see Lemma 8 below), we will encounter a marking that is larger or equal to a marking encountered before (line 6). Then, the algorithm can decide: if the cycle between these two markings would be repeated forever, it still would not enable any other transition at any point along the cycle (line 9). If that requirement hold, then M is part of a non-decreasing livelock.

Lemma 8. *Algorithm 1 decides after finitely many steps whether marking M is part of a non-decreasing livelock.*

Proof. Assume M is part of a non-decreasing livelock. Then, there exists one enabled transition t in line 3. This yields an infinite sequence of markings $\mu = \langle M_1, \dots \rangle$. By Dickson's Lemma [7], there exist indices $k < \ell \in \mathbb{N}$ with $M_k \trianglelefteq M_\ell$, which we detect in line 6. The sequence of transitions fired between M_k and M_ℓ can be fired again, starting in M_ℓ instead of M_k. This yields a cycle that is traversed infinitely often. Let $\Delta = M_\ell \setminus M_k$ be the increase in markings that we gain after traversing the cycle once. Then the markings that we visit after M_ℓ are all given by $M_i \uplus \biguplus_{j=1}^{\alpha} \Delta$, with $k \le i < \ell$ and $\alpha \in \mathbb{N}$ being the number of times we traverse the cycle. Since we are in a non-decreasing livelock, there is only one marking enabled for all $k \le i < \ell$ and $\alpha \in \mathbb{N}$, hence line 9 returns true.

Assume that M is not part of a non-decreasing livelock., and that line 6 is true at some point. Otherwise, the while loop terminates because there is not

exactly one enabled transition, and the algorithm returns false. After detecting $k < \ell \in \mathbb{N}$ with $M_k \trianglelefteq M_\ell$, the transitions fired between M_k and M_ℓ can be fired infinitely often. Since we are not in a non-decreasing livelock, there must be another enabled transition at some point. Such a transition is enabled in marking $M_i \uplus \biguplus_{j=1}^{\alpha} \Delta$ with $k \leq i < \ell$ and $\alpha \in \mathbb{N}$. Hence, $w(t) > 0 \wedge {}^\bullet t \trianglelefteq M_i \uplus \biguplus_{j=1}^{\infty} \Delta$ for more than one transition $t \in T$, and the algorithm returns false in line 9. □

3.2 Unbounded SLPN

We recall the standard notion of *boundedness*:

Definition 9 (Bounded SLPN). *Let $(P, T, F, M_0, \Sigma, \lambda, w)$ be an SLPN, and let $\bar{M}$ be the set of all reachable markings from M_0. If $\bar{M}$ is finite, then the SLPN is called* bounded.

Worded differently, a place is bounded if there exists a k such that every reachable marking has at most k tokens in that place. An SLPN is bounded if all of its places are bounded. An unbounded marking can only be "reached" through the execution of infinitely many transitions. Since T is finite, for every unbounded net, there must be a sequence of transitions that can be repeated infinitely often and that puts infinitely many tokens into a place:

Definition 10 (Unbounded (silent) cycle). *Let $(P, T, F, M_0, \Sigma, \lambda, w)$ be an SLPN, and let $\bar{M}$ be the set of all reachable markings from M_0. Let $M_i \in \bar{M}$ be a reachable marking and let $\langle t_1 \dots t_n \rangle$ be a sequence of transitions such that $M_i \xrightarrow{t_1} \dots \xrightarrow{t_n} M_j$. If $\forall_{p \in P} M_i(p) \leq M_j(p)$ and $\exists_{p \in P} M_i(p) < M_j(p)$, then $(M_i, \langle t_1 \dots t_n \rangle)$ is an* unbounded cycle. *The cycle is an* unbounded silent cycle *if all of the transitions $t_1 \dots t_n$ are silent.*

An SLPN with an unbounded cycle is unbounded: this cycle can be repeated, causing a place to have unboundedly many tokens. Conversely, every unbounded SLPN has at least one unbounded cycle. The proof is analogous to [10].

4 Q-State Space of an SLPN

We define a q-state space over the behaviour of an SLPN, introduce a method to compute it, and show how the q-state space can be explored to address our Problem. The q-state space yields a stochastic deterministic automaton, that is, it is fully deterministic, but may have an unbounded number of states.

4.1 Q-State Semantics

A *probabilistic state* (q-state) Q is a q-set of markings, describing the distribution over markings the net *could* be in. We use exponents to denote the marking probabilities within a q-state, e.g., $[[p_1]^{0.5}, [p_2]^{0.5}]$ indicates a 0.5 probability of being in either marking $[p_1]$ or $[p_2]$. The initial q-state of a net with initial marking M_0 is $Q_0 = [M_0^1]$. In our example, it is $[[p_0]]$.

SLPN transition t is *enabled* in q-state Q if Q has an enabling marking for t with positive probability: $\exists_{M \in \widetilde{Q}} {}^\bullet t \trianglelefteq M \wedge w(t) > 0$. A transition from a q-state Q to a q-state Q' with activity a, denoted with $Q \overset{a}{\underset{p}{\rightsquigarrow}} Q'$, indicates that from at least one of the markings in $\widetilde{Q}$, we can take zero or more silent transitions in the SLPN, followed by a single firing of a transition labelled with a. After the firing of the transition labelled with a, no more silent transitions must be executed. The resulting Q' is the distribution over markings we may end up in, and p is the probability that a is executed in Q.

While an SLPN can only deterministically terminate – in a deadlock –, a q-state space can terminate stochastically, if one of its markings is a deadlock. In a q-state Q, execution can terminate with probability $p_\perp(Q)$.

Finally, a q-state Q may partially or completely be a silent livelock. That is, for some of its markings, it may not be possible to eventually terminate or execute an activity. The sum probability of these markings in Q is called the *silent livelock probability* which we denote with $\bullet(Q)$. Obviously, it holds that $p_\perp(Q) + \bullet(Q) + \sum_{Q \overset{a}{\underset{p}{\rightsquigarrow}} Q'} p = 1$. The *livelock probability* $\mathrm{p}_\circ(Q)$ is the probability that the SLPN is in a livelock in Q: $\mathrm{p}_\circ(Q) = \sum_{M \in \widetilde{Q} \wedge \bigcirc(M)} Q(M)$.

4.2 Q-State Transitions

We compute the outgoing edges of a q-state Q in four steps, detailed next. (1) We construct a Markov chain with its states representing all markings that can be reached from any marking of $\widetilde{Q}$ by executing any number of silent transitions, followed by one labelled transition. (2) We transform this Markov chain into an absorbing Markov chain. (3) We solve the Markov chain to obtain the likelihoods of each next marking. (4) From the solved Markov chain, we obtain the outgoing transitions of Q, as well as the successor q-states they lead to.

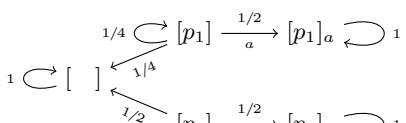
Constructing the Markov chain. From a set of markings $\widetilde{Q}$ we create a Markov chain, where states represent the markings that can be reached in the SLPN, and transitions represent the transitions of the SLPN. Silent transitions of the SLPN are traversed without limitation, however if a labelled transition is encountered, it is taken but the reached marking is not explored further.

We define the states S of the Markov chain by using an auxiliary set S', which allows us to follow silent transitions until we have fired a labelled transition, and then stop. That is, let S and S' be the smallest sets such that: (1) all markings of $\widetilde{Q}$ are in S and S'; (2) all markings reachable from a state in S' by firing silent transitions are in S and S'; and (3) all markings reachable from a state in S' by firing a transition labelled with an activity $a \in \Sigma$ are in S (we annotate these markings with activity label a, and refer to them as *after-activity states*).

$$M \in \widetilde{Q} \Rightarrow (M \in S \wedge M \in S') \tag{1}$$

$$(M \in S' \wedge M \xrightarrow{\tau} M') \Rightarrow (M' \in S \wedge M' \in S') \tag{2}$$

$$(M \in S' \wedge M \xrightarrow{a} M') \Rightarrow M'_a \in S \qquad \textit{after-activity state} \tag{3}$$



	$[p_1]_a$	$[\,]$	$[p_2]_a$	$[p_1]$	$[p_2]$
$[p_1]_a$	1	0	0	0	0
$[\,]$	0	1	0	0	0
$[p_2]_a$	0	0	1	0	0
$[p_1]$	1/2	1/4	0	1/4	0
$[p_2]$	0	1/2	1/2	0	0

Fig. 2: Markov chain and its matrix of marking set $\widetilde{Q} = \{[p_1], [p_2]\}$ of our example from Figure 1.

The probability function p', denoting the edges of the Markov chain, is constructed according to the mentioned SLPN transitions, with the probabilities they have in the SLPN ((4) and (5)). Additionally, the after-activity states (6) and deadlock markings (7) become absorbing states:

$$\begin{pmatrix} M \in S \wedge \\ M \xrightarrow{\tau} M' \end{pmatrix} \Rightarrow p'(M, M') = \sum_{{}^\bullet t \subseteq M \wedge \lambda(t) = \tau \wedge M \uplus t^\bullet \setminus {}^\bullet t = M'} p(M, t) \tag{4}$$

$$\begin{pmatrix} M \in S \wedge \\ M \xrightarrow{a} M'_a \end{pmatrix} \Rightarrow p'(M, M'_a) = \sum_{{}^\bullet t \subseteq M \wedge \lambda(t) = a \wedge M \uplus t^\bullet \setminus {}^\bullet t = M'} p(M, t) \tag{5}$$

$$M_a \in S \Rightarrow p'(M_a, M_a) = 1 \tag{6}$$

$$\begin{pmatrix} M \in S \wedge \\ \forall_t {}^\bullet t \not\subseteq M \end{pmatrix} \Rightarrow p'(M, M) = 1 \tag{7}$$

For our example from Figure 1 (the second q-state), the Markov chain with its transition matrix $\mathcal{T}$ is shown in Figure 2.

Note that in case an unbounded silent cycle (Definition 10) is encountered, Equation (2) covers an unbounded number of states, and a practical computation would not terminate, as the Markov chain has infinitely many states. Even though the Markov chain is still well defined, we leave it for future work to establish whether such Markov chains can be solved.

Making the Markov chain absorbing. For the Markov chain to be solved, it needs to be absorbing. The after-activity states (6) and deadlock-marking states (7) are absorbing by construction. Furthermore, if the SLPN is free of silent livelocks (Definition 6), then from every state a deadlock-marking state (7) or an after-activity state (3) can be reached, thus the Markov chain is absorbing.

However, if a silent livelock can be reached, then in the Markov chain there is a state from which neither an after-activity state (3) nor a deadlock-marking state (7) can be reached. Algorithm 2 shows a procedure to identify and address these states. First, the absorbing states are identified through their self-loop probability of 1. Second, a backward search is performed from the absorbing states to identify all states that can reach an absorbing state (line 3). Each state that is not encountered in this search can thus not reach an absorbing state, which means that it is a state that is part of a livelock of silent transitions. As such, we mark it as a *silent-livelock state* for later use, and all its outgoing

Algorithm 2 Make a Markov chain absorbing

```
1: procedure ABSORBISE(Markov chain (S, p))
2:     X ← absorbing states of S
3:     while X changes do                ▷ search states that can reach an absorbing state
       X ← X ∪ {s ∈ S | s′ ∈ X ∧ p(s, s′) > 0} end while
4:     for s ∈ S \ X do                  ▷ make all other states absorbing
5:         mark s as a livelock state
6:         p(s, s) ← 1
7:         for s′ ∈ S \ {s} do p(s, s′) ← 0 end for
8:     end for
9:     return (S, p)
10: end procedure
```

transitions are removed and replaced with a single self-transition, which makes the state absorbing (line 6). Intuitively, this changes the behaviour of the livelock, however this obviously does not change the behaviour represented by the Markov chain. Applying this procedure to all such states ensures that every state can reach an absorbing state, which makes the Markov chain absorbing.

Solving the Markov chain. We solve the absorbing Markov chain using a standard procedure [12]. First, $\mathcal{T}$ is put in canonical form: all rows and columns of absorbing markings are listed first. Then, $\mathcal{T}$ has the following shape: $\begin{bmatrix} I & 0 \\ \mathcal{A} & \mathcal{B} \end{bmatrix}$, where I is an identity matrix and 0 is a matrix of zeroes. Second, the fundamental matrix $\mathcal{F} = (I - \mathcal{B})^{-1}$ is computed. As the Markov chain is absorbing, the existence of the matrix inverse is guaranteed [12]. Each entry $\mathcal{F}_{i,j}$ denotes the average number of times we visit marking j if we start in marking i before moving to an absorbing state. Then, $\mathcal{P} = \mathcal{F}\mathcal{A}$ is computed, which we substitute into $\begin{bmatrix} I & 0 \\ \mathcal{P} & 0 \end{bmatrix}$ to obtain the matrix $\mathcal{T}^\infty$. Given states of marking M, M', $\mathcal{T}^\infty_{M,M'}$ is the probability that the net ends up in M' when starting in M. For our example:

$$\mathcal{A} = \begin{array}{c} \\ [p_1] \\ [p_2] \end{array} \begin{array}{c} \begin{array}{ccc} [p_1]_a & [\,] & [p_2]_a \end{array} \\ \begin{bmatrix} 1/2 & 1/4 & 0 \\ 0 & 1/2 & 1/2 \end{bmatrix} \end{array} \quad \mathcal{B} = \begin{array}{c} \\ [p_1] \\ [p_2] \end{array} \begin{array}{c} \begin{array}{cc} [p_1] & [p_2] \end{array} \\ \begin{bmatrix} 1/4 & 0 \\ 0 & 0 \end{bmatrix} \end{array} \quad \mathcal{F} = (I - \mathcal{B})^{-1} = \begin{array}{c} \\ [p_1] \\ [p_2] \end{array} \begin{array}{c} \begin{array}{cc} [p_1] & [p_2] \end{array} \\ \begin{bmatrix} 4/3 & 0 \\ 0 & 1 \end{bmatrix} \end{array}$$

$$\mathcal{P} = \mathcal{F}\mathcal{A} = \begin{array}{c} \\ [p_1] \\ [p_2] \end{array} \begin{array}{c} \begin{array}{ccc} [p_1]_a & [\,] & [p_2]_a \end{array} \\ \begin{bmatrix} 2/3 & 1/3 & 0 \\ 0 & 1/2 & 1/2 \end{bmatrix} \end{array} \qquad \mathcal{T}^\infty = \begin{array}{c} \\ [p_1]_a \\ [\,] \\ [p_2]_a \\ [p_1] \\ [p_2] \end{array} \begin{array}{c} \begin{array}{ccccc} [p_1]_a & [\,] & [p_2]_a & [p_1] & [p_2] \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 2/3 & 1/3 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 \end{bmatrix} \end{array}$$

Extracting the next q-states. Next, we multiply the incidence vector $\boldsymbol{Q}$ of q-state Q, with added zeroes where necessary, with $\mathcal{T}^\infty$, to obtain a distribution

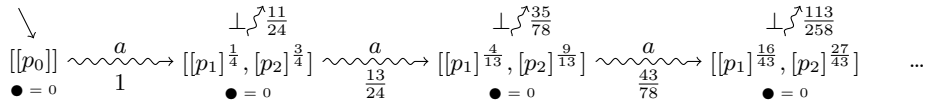


Fig. 3: Q-state space of our example SLPN from Figure 1.

over states $\mathcal{N}$ that we end up in if we proceed from Q. For our example:

$$\boldsymbol{Q} = \begin{matrix} \end{matrix} \begin{matrix} [p_1]_a & [\,] & [p_2]_a & [p_1] & [p_2] \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 1/4 & 3/4 \end{bmatrix}$$

$$\mathcal{N} = \boldsymbol{Q}\mathcal{T}^{\infty} = \begin{matrix} \end{matrix} \begin{matrix} [p_1]_a & [\,] & [p_2]_a & [p_1] & [p_2] \end{matrix} \\ \begin{bmatrix} 1/6 & 11/24 & 3/8 & 0 & 0 \end{bmatrix}$$

There are two after-activity states that indicate that an a was executed ($[p_1]_a$ and $[p_2]_a$). The probability that we execute an a in Q is the sum of these probabilities ($13/24$). If we execute an a, then by definition of the q-state space, we first fire any number of silent transitions in the SLPN, followed by a single firing of a transition with the label a. The markings we may then end up in are thus, in our example, $[[p_1]^{1/6}, [p_2]^{3/8}$. The sum of these markings is the probability that we execute an a; to obtain the next q-state Q', the last step is to normalise this into a distribution: in our example, $Q \overset{a}{\underset{13/24}{\rightsquigarrow}} [[p_1]^{\frac{4}{13}}, [p_2]^{\frac{9}{13}}]$.

Similarly, the probability to terminate from Q is the sum of probabilities of all markings that are deadlocks in $\mathcal{N}$. In our example, this is $p_{\perp}(Q) = 11/24$.

Finally, the probability to get stuck in a silent livelock in Q, denoted with $\bullet(Q)$, is the sum of probabilities of the states that were marked as silent-livelock states in Algorithm 2. In our example, this is 0.

The effect of firing activity a can be summarized removing unnecessary rows and columns from $\mathcal{P}$, obtaining a smaller matrix $\mathcal{P}'_a$. In our example, we get:

$$\mathcal{P}'_a = \begin{matrix} \end{matrix} \begin{matrix} [p_1]_a & [p_2]_a \end{matrix} \\ \begin{matrix} [p_1] \\ [p_2] \end{matrix} \begin{bmatrix} 2/3 & 0 \\ 0 & 1/2 \end{bmatrix}$$

This method, for any SLPN, defines a state space of q-states (that is, an SDA), and provides a way to traverse this space by computing states incrementally. For our example, the first states of the q-state space are shown in Figure 3. In Section 5, we use this method to answer probabilistic queries on SLPNs. But first, we discuss the computability of the q-state space.

4.3 Computability

Next, we prove the computability of the procedure to traverse the q-state space. Notice that due to the Markov chain construction, we can only do this for SLPNs without unbounded silent cycles.

Lemma 11. *Let N be an SLPN without unbounded silent cycles, and let Q be a q-state that is reachable from $[M_0]$. Then, the successors of Q can be computed.*

Proof. Prove by induction that $\widetilde{Q}$ is finite. Base case: the initial q-state contains one marking (M_0), thus is finite. Induction step: assume that $\widetilde{Q}_i$ is finite. Construction of the Markov chain (Section 4.2) follows a state space exploration of markings of the SLPN, and stops at every labelled transition (Equation 3). As N contains no unbounded silent cycles, the Markov chain has a bounded number of states, and thus any successor $\widetilde{Q}_j$ of $\widetilde{Q}_i$ is finite. Through the method of this section, computing the successors of any reachable q-state $\widetilde{Q}$ terminates. □

5 Answering probabilistic queries

We now use the q-state space to answer probabilistic queries in the sense of our Problem: given an SLPN and a stopping criterion $\varphi\colon (\Sigma^*)^{\mathbb{R}} \times \mathbb{R} \to \mathbb{R}$, we introduce an algorithm that adds the traces of the SLPN in descending order of likelihood, until either all traces have been added or φ returns true. Algorithm 3 shows the procedure. Intuitively, it performs a search of all prefixes supported by the model, in order of *potential probability*, that is, an upper bound on how much probability mass the prefix may contribute to the final result.

We define the potential probability of a prefix trace as $\rho(\sigma) = p_\sigma * (1 - \mathrm{p}_\circ(\sigma))$, in which we with $\mathrm{p}_\circ(\sigma)$ denote the probability that the last q-state after executing σ is part of a livelock. Moreover, p_σ is the probability that $\sigma = \langle a_1 \dots a_k \rangle$ is executed: $p_\sigma = \prod_{a_i \in \sigma \wedge Q_{i-1} \overset{a_i}{\underset{p}{\rightsquigarrow}} Q_i} p$. The prefix with the highest potential probability is expanded from its last q-state Q by:

- If the prefix can terminate with probability $p_\perp(Q)$, the corresponding trace is added to the queue with its probability as priority (line 8). This ensures that final traces are processed in order of their probability (Lemma 13).
- For each outgoing edge $Q \overset{a}{\underset{p}{\rightsquigarrow}} Q'$ that does not lead to a certain livelock (line 12), the prefix is expanded to include a (line 13). The potential probability of $\sigma \cdot \langle a \rangle$ is the product of the probability to execute σ, the probability p of executing a in Q, and the probability that Q' is not in a livelock.

As soon as φ returns true or an error (lines 5), or all traces have been traversed (line 20), the procedure ends.

Next, we prove that traces are added in order of increasing probability (Lemma 13) and that the algorithm terminates (Theorem 16). Towards Lemma 13, we first prove that potentials are non-increasing and an upper bound on trace probabilities (Lemma 12). As a prefix of a trace uniquely identifies a q-state, we lift the functions $\mathrm{p}_\circ$ and $p_\perp$ to prefixes of traces. Recall that $\mathcal{L}(\sigma) = p_\sigma * p_\perp(\sigma)$.

Lemma 12. *Let N be an SLPN and σ be a trace of N. Then, for all traces σ' in N of which σ is a prefix, it holds that $\mathcal{L}(\sigma') \leq \rho(\sigma') \leq \rho(\sigma)$.*

Proof. Note that $\rho(\sigma') = p_{\sigma'} * (1 - \mathrm{p}_\circ(\sigma'))$. As "not being in a livelock" includes termination, $p_\perp(\sigma') \leq 1 - \mathrm{p}_\circ(\sigma')$ holds, which proves the first inequality. For the second inequality, we show that $({}^{p_{\sigma'}}/_{p_\sigma}) * (1 - \mathrm{p}_\circ(\sigma')) \leq 1 - \mathrm{p}_\circ(\sigma)$. Let $\sigma' = \sigma * \langle a_1 \dots a_k \rangle$ and note that ${}^{p(\sigma')}/_{p(\sigma)}$ is the probability of executing $a_1 \dots a_k$ after σ. The value $({}^{p_{\sigma'}}/_{p_\sigma}) * (1 - \mathrm{p}_\circ(\sigma'))$ is the probability to execute $a_1 \dots a_k$

Algorithm 3 Answer a probabilistic query

```
 1: procedure QUERY(SLPN (P, T, F, M_0, Σ, λ, w), stopping criterion φ)
 2:     R ← {(prefix, ⟨ ⟩) with potential probability 1}                ▷ queue
 3:     S ← []                                                          ▷ result
 4:     while remove (Y, σ) with highest priority v from R do
 5:         if φ(S, v) then return S end if
 6:         if Y = prefix then                                  ▷ prefix at head of queue
 7:             Q ← last q-state of σ
 8:             if p_⊥(Q) > 0 then                              ▷ Q is final; trace found
 9:                 add (trace, σ) to R with priority p_σ * p_⊥(Q)
10:             end if
11:             for Q ⇝^a Q' with probability p_a > 0 do     ▷ using method of Section 4
12:                 if ∃_{M∈Q̃'} ¬○(M) then                  ▷ Q has a non-livelocked marking
13:                     add (prefix, σ · ⟨a⟩) to R with priority p_σ * p_a * (1 − p_∘(Q'))
14:                 end if
15:             end for
16:         else                                              ▷ completed trace at head of queue
17:             add (σ, v) to S
18:         end if
19:     end while
20:     return S
21: end procedure
```

and not be in a livelock afterwards. This is only possible if we are not already in a livelock after σ. Hence this probability is included in $1 - \mathrm{p}_\circ(\sigma)$, which proves the second inequality. □

Lemma 13 (Algorithm 3 returns traces in order of probability). *Let N be an SLPN. Let σ_1 be a trace that is added to S before a trace σ_2 in Algorithm 3. Then, $\mathcal{L}(\sigma_1) \geq \mathcal{L}(\sigma_2)$.*

Proof. When σ_1 is added to S, it has the highest priority in R, being $\mathcal{L}(\sigma_1)$.

- If σ_2 is already in the queue as a trace with priority $\mathcal{L}(\sigma_2)$, then by the priority queue it holds that $\mathcal{L}(\sigma_1) \geq \mathcal{L}(\sigma_2)$.
- If σ_2 is not in the queue yet, then there is some σ_3 in the queue that is a prefix of σ_2 with potential $\rho(\sigma_3) \leq \mathcal{L}(\sigma_1)$. By Lemma 12, $\mathcal{L}(\sigma_2) \leq \rho(\sigma_3)$, and thus $\mathcal{L}(\sigma_1) \geq \mathcal{L}(\sigma_2)$.
- If σ_2 is already in the queue as a prefix, the previous case applies as σ_2 is also a prefix of itself. □

Towards the proof of termination (Theorem 16), we use a conjecture that states that for every infinite sequence of prefixes, the potential converges to zero. The only only way the potential could not converge to zero, is if both the prefix probability p_σ does not converge to zero, and the livelock probability $\mathrm{p}_\circ(\sigma)$ does not converge to one. However, if p_σ does not converge to zero, then the probability of firing the next activity converges to one. We conjecture that this is only possible if the probability of being in a livelock converges to one.

Conjecture 14. For all $i \in \mathbb{N}$, let $\sigma_i \in \Sigma^i$ be a prefix trace such that the potential $\rho(\sigma_i)$ is maximal among all prefix traces of length i. The sequence of potentials $(\rho(\sigma_i))_{i \in \mathbb{N}}$ converges to zero.

Given the conjecture, we prove that finitely many prefixes are added in front of any trace on the queue.

Lemma 15. *Let N be an SLPN and σ be a trace of N. Let φ always return false. If Conjecture 14 holds, then eventually σ gets added to S.*

Proof. According to Conjecture 14, there exists an $i \in \mathbb{N}$ with $\rho(\tilde{\sigma}) < \mathcal{L}(\sigma)$ for all prefix traces $\tilde{\sigma}$ of length at least i. Since Lemma 12 implies $\mathcal{L}(\sigma) \leq \rho(\sigma')$ for all prefixes σ' of σ, we consider every prefix of σ at the top of the priority queue before considering any prefix trace of length i or longer. In particular, σ will be added as a trace to the queue and will be at the top of the queue before any trace of length at least i is considered. Since there are only finitely many traces shorter than i, we add σ to S after finitely many steps. □

Theorem 16 (Algorithm 3 terminates). *Let $N = (P, T, F, M_0, \Sigma, \lambda, w)$ be an SLPN without unbounded silent cycles. Let φ be a stopping function that in context of Algorithm 3 returns true after a bounded number of calls. Then, if Conjecture 14 holds, Algorithm 3 applied to N terminates.*

Proof. As line 11 terminates by Lemma 11, Σ is bounded, and line 12 is decidable by [9, thms 3.2.25 & 3.2.26], all lines in isolation terminate. Next, towards contradiction, assume that Algorithm 3 does not terminate, which means that R never gets empty. Still, by our assumption on φ, a bounded number of traces gets added to S. Then, by line 12, some trace σ of N never reaches the head of the queue, which contradicts Lemma 15. Hence, Algorithm 3 terminates. □

Next, we provide three examples of stopping conditions φ, each corresponding to a particular type of probabilistic query.

5.1 Most-likely traces

The first type of probability query asks for a given number (n) of most-likely traces. The corresponding stopping condition returns whether n has been reached:

$$\varphi_{\text{most likely}}(S, v) = |S| \geq n \tag{8}$$

Obviously, $\varphi_{\text{most likely}}$ returns true at a bounded number of traces (n). Hence by Theorem 16, if Conjecture 14 holds, Algorithm 3 with $\varphi_{\text{most likely}}$ terminates.

5.2 Probability mass

The second type of probabilistic query is to find the most likely traces that together cover a probability mass larger than a given f. In two cases, it is not possible to obtain such a set S. First, if the probabilty that the SLPN ends up

in a livelock from the initial marking is ψ, and f is larger than $1 - \psi$, then the query is not satisfiable (9a). Second, if the SLPN contains an unbounded number of traces (represented by ℓ) and the requested probability mass f is equal to the livelock probability, the query is unsatisfiable as well (9b).

Otherwise, the corresponding stopping function φ_{cover} returns whether the requested mass f has been gathered (9c):

$$\varphi_{\text{cover}}(S, v) = \begin{cases} \text{query not satisfiable} & \text{if } f > 1 - \psi & \text{(9a)} \\ \text{query not satisfiable} & \text{if } f = 1 - \psi \wedge \ell & \text{(9b)} \\ f \leq \sum\limits_{(p,\sigma) \in S} p & \text{otherwise} & \text{(9c)} \end{cases}$$

The measure ψ (livelock probability) could, for bounded SLPNs, be computed using Markov chain techniques similar to [19]. The measure ℓ (loops) can be computed, for bounded SLPNs, by waiting for a trace to be added to S that has two q-states that contain the same marking. For both ψ and ℓ, detection on unbounded models remains future work.

As the SLPN has a total probability mass of $1-\psi$, it holds that φ_{cover} returns true at a bounded number of traces. Thus, if ψ and ℓ are known, by Theorem 16, Algorithm 3 with φ_{cover} terminates.

5.3 Minimum probability

The third type of probabilistic query is to obtain all traces that have a probability equal to or higher than a given f. If $f > 0$, the SLPN has at most $1/f$ traces with a probability $\geq f$, and the criterion returns true if $v < f$, that is, when the potential v of the last-visited prefix cannot lead to a trace with a probability $\geq f$ anymore. Otherwise, their combined probability would exceed 1 (10b). If the number of traces in the SLPN is unbounded, then there are an unbounded number of traces with a probability ≥ 0. Therefore, if $f = 0$ and ℓ hold, then the query has no answer (10a).

$$\varphi_{\text{minimum probability}}(S, v) = \begin{cases} \text{query not satisfiable} & \text{if } f = 0 \wedge \ell & \text{(10a)} \\ v < f & \text{otherwise} & \text{(10b)} \end{cases}$$

Consequently, if Conjecture 14 holds, Algorithm 3 with $\varphi_{\text{minimum probability}}$ terminates by Theorem 16. By Lemma 13, after termination S contains the traces of the SLPN with probability greater than f.

However, for $\varphi_{\text{minimum probability}}$, we can weaken the check for livelocks on line 12 to non-decreasing livelocks (Definition 7). The more times a decreasing livelock is executed, the lower the probability of the partial path, thereby steadily decreasing its probability, until eventually f is reached. Hence, we do not need to explicitly check for decreasing livelocks. This argument does not hold for non-decreasing livelocks, which do not decrease in probability, thus line 12 can be replaced with a (cheaper) check for non-decreasing livelocks, as in Algorithm 1. We refer to this adapted algorithm as Algorithm 3'.

Lemma 17 (Algorithm 3' terminates with $\varphi_{\text{minimum probability}}$). *Let N be an SLPN without unbounded silent cycles. Then, if Conjecture 14 holds, Algorithm 3' applied to N terminates.*

Proof. Algorithm 1 visits each transition at most once, thus the replaced line 12 in isolation terminates. With reference to the proof of Theorem 16, left to prove: eventually, p drops below f. Assume towards contradiction that N has no non-decreasing livelock, $f > 0$ and that p never drops below f. Then, there exist infinitely many prefixes of probability $\geq f$. A prefix can be expanded with either (i) termination, (ii) a choice between multiple transitions, (iii) a forced single enabled transition, and (iv) a silent livelock. Cases (i), (ii) and (iv) can only be applied a bounded number of times, so in order to have infinitely many prefixes, case (iii) must be applicable infinitely many times, which can only happen infinitely many times in a non-decreasing livelock, which is a contradiction. If $f = 0$ and $\neg\ell$, the proof is similar. Hence, Algorithm 3' terminates. □

6 Evaluation

We now evaluate our approach for answering probabilistic queries on SLPNs: we discuss its implementation, and evaluate its feasibility and applicability.

6.1 Implementation

The method of Section 5 has been implemented in the open-source stochastic process mining framework Ebi (`https://ebitools.org` [17]). The three types probabilistic queries have been implemented: most likely traces ($\varphi_{\text{most likely}}$) and probability mass ($\varphi_{\text{cover}}$) have been implemented without livelock check but with non-decreasing livelock check, while minimum-probability ($\varphi_{\text{minimum probability}}$) has been fully implemented.

Given the sequential nature of the computation of q-states and the potentially low probabilities of traces (we encountered fractional numbers of 18 000 divided by 19 000 bits), IEEE double precision floating point numbers may not provide the necessary range and accuracy. Therefore, the implementation uses infinite-precision fractions throughout, though a user can disable this and revert back to double precision for run time considerations. The experiments reported hereafter were all performed with exact arithmetic.

Conceptually, the methods of Section 5 support application to any stochastic modelling formalism (with the exception of the livelock check). Subsequently, the implementation supports any modelling formalism with stochastic semantics, which currently includes SLPNs, SDFAs, finite stochastic languages and event logs (though direct, more efficient, special cases apply for the latter two).

6.2 Feasibility

To test the feasibility of our implementation, we apply it to all 522 XES logs listed by the IEEE Task Force on Process Mining (`https://www.tf-pm.org/`

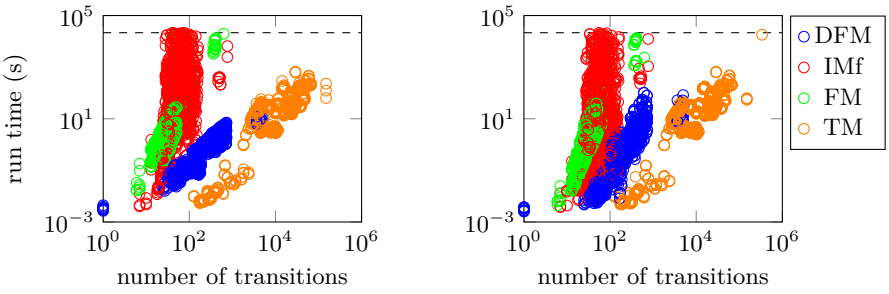
Table 1: Details of the experimental set-up.

(a) Control-flow discovery.

Inductive Miner - infrequent (0.8) [16] transformed to a labelled Petri net	IMf
Directly Follows Model Miner (0.8) [21] transformed to a labelled Petri net	DFM
flower miner: allow for any behaviour	FM
trace model: a model with the language of the log	TM

(b) Stochastic discovery.

uniform: assign each transition a weight of 1	UNI
occurrence: assign each transition the occurrence of its label, and 1 to silent transitions [3]	OCC
alignments: perform alignments and assign each transition its occurrences [3]	ALI



(a) Minimum probability. (b) Most-likely traces.

Fig. 4: Results of the feasibility experiment. Dashed lines indicate the 6h timeout.

`resources/logs`, accessed 7-5-2024). To resemble real-life use cases, we first apply 4 process discovery techniques to obtain a variety of control-flow models for each log, after which each of these models is enriched with a stochastic perspective by 3 stochastic process discovery techniques. More details are shown in Table 1. To the resulting 6 253 stochastic process models[5], we apply the three types of probabilistic queries:

- We extract the traces that have a 0.1% or higher probability, and measure the time this takes with a timeout of 6 hours;
- We extract the 10 most likely traces, and measure the time this takes with a timeout of 6 hours;
- For coverage, we take a different approach as coverage is highly dependent on the model's number of traces. We extract the most-likely traces that together cover 99.999% of the probability mass, with a timeout of 1 minute. We measure the time that this takes, and the number of traces reported in that 1 minute.

All measures were taken on an AMD 5964wx CPU with 256GB of RAM available. The results are shown in Figure 4. The minimum probability querying timed out in 151 cases, while the most-likely trace querying timed out in 116 cases.

The different discovery techniques have different characteristics: the cheapest to compute were DFM and FM, as their models have a small marking space, due to the models being deterministic and having a limited number of silent

[5] The ALI stochastic process discovery ran out of RAM for 11 models, thus these were not further considered.

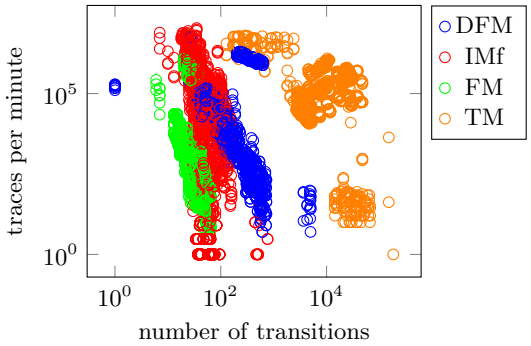


Fig. 5: Results of the coverage feasibility experiment.

transitions. Hence, all q-states of DFM-models have only one marking, and the q-states of FM models have at most two markings.

TM models have very large q-states, as initially a transition is enabled for every trace of the event log. The q-states get smaller whenever a transition is fired, and as such, TM models utilise the q-state space extensively. For complex models with lots of activities, the q-states get smaller quite quickly, so for most large models, there is an initial expensive q-state, but subsequent q-states get small quickly; TM-models with 10^6 transitions could still be handled in a few minutes. However, the worst case is IMf, which tends to introduce silent transitions, which may involve high numbers of large q-states.

We conclude that our method is feasible in practical cases. Nevertheless, we observed run times of over a day for the most complex models, which involved over a million transitions and used more than 150GB of RAM, though these were clearly exceptions, as smaller models took a few minutes or seconds.

6.3 Applicability

To illustrate the applicability of our approach, we consider an analyst tasked with analysing a loan application process (BPIC2012). A process model was created using IMf [16], and annotated with the occurrence miner [3]. To get an idea of the quality of the model, we can compute the uEMSC [22] value, which with 0.015 is rather low. To investigate the quality of the model further, we apply our approach to obtain the 5 most-likely traces from this stochastic model:

Trace	probability in model	probability in log
⟨`submitted`, `partlysubmitted`, `afhandelen leads`⟩	0.139	0
⟨`submitted`, `partlysubmitted`⟩	0.099	0
⟨`submitted`, `partlysubmitted`, `preaccepted` ⟩	0.062	0
⟨`submitted`, `partlysubmitted`, `afhandelen leads`, `afhandelen leads`⟩	0.018	0
⟨`submitted`, `partlysubmitted`, `completeren aanvraag`⟩	0.015	0

The list confirms that the model does not describe the log well, as of the 5 most likely traces *none* appears in the event log *at all*. As such, the analyst should proceed with caution with this stochastic model and perhaps apply a different discovery approach.

7 Related Work

Stochastic labelled Petri nets (SLPNs) have been used before to describe stochastic business processes: they can be discovered [18] and their conformance to an event log or another stochastic model can be computed [15,19]. To compute some types of conformance, trace queries must be answered to obtain the probability that the LSPN will generate a given trace [22]. Silent steps and repeated labels make such queries non-trivial to answer. In [19], it is shown that trace queries can be answered analytically over bounded LSPNs, by extracting a Markov chain from the LSPN (considering transitions), and by defining a suitable notion of cross-product with a given temporal specification (expressed over labels). Unfortunately, the approach in [19] cannot be employed to answer probability queries as considered in this paper. Intuitively, this is because probabilities are measured over traces, and one needs to associate trace prefixes to probabilities, incrementally updating them towards full traces. This calls for a state space where silent transitions have been removed (retaining their probabilities), and where transitions with the same label are suitably merged, while at the same time retaining the information on which distinct states they lead to, each with each probability. This is what our q-state space captures (cf. Section 4).

Generalised stochastic Petri nets (GSPN [4,27]) extend SLPNs by distinguishing between immediate and timed transitions, where immediate transitions have priority over timed transitions. Notably, if one is interested only in traces and their probabilities, immediate and timed transitions behave homogeneously, due to the fact that in GSPNs timed transitions have exponentially distributed durations. Our results seamlessly carries over to GSPNs, which do not support silent transitions, following the same line of reasoning from [19].

The construction of the q-state space we provide resembles the construction of a (a possibly infinite-state) stochastic deterministic automaton. Interestingly, this tightly corresponds to the (deterministic versions of) automata models introduced in the seminal works [28,29]. A property discussed in [28] is that stochastic non-deterministic finite automata (SNFAs, called PFAs in [28]) are strictly more expressive than their deterministic counterpart, which corresponds to our q-state spaces being either finite or infinite. The addition of silent transitions to SNFAs (called λ-PFAs in [28]) does not increase their expressivity. Nevertheless, on λ-PFAs, many problems, including the probabilistic queries we address in this paper, become intractable and are NP-hard. This could be exploited to provide complexity results for the probability queries studied in this paper.

Hidden Markov models (HMMs) are equivalent to SNFAs, with the exception that in contrast to SNFAs, HMMs cannot express the empty trace [29]. Furthermore, HMMs are typically used to study state-based properties, whereas within process mining, we are interested in transitions and their sequencing. A more elaborate discussion on the relation between HMMs and SDFAs can be found in [29]. At the surface, Markov decision processes (MDPs) [25] appear closer to SDFAs, due to their focus on actions and decision making. However, they are substantially different: in MDPs transition choices are not stochastic but purely non-deterministic, while the state in which the systems evolves in after picking a

transition is probabilistic. Due to the state-based nature of Markov models, the problem of identifying the most-likely trace (possibly involving silent transitions) is not a typical focus of Markov analysis [13], and to the best of our knowledge, has not been addressed, in particular as we consider infinite state spaces. Nevertheless, the methods described in this paper apply directly to Markov models with labelled transitions.

Another type of stochastic automaton (R-SDFA) is defined in [26]. This model is closer to MDPs, but differs from our q-state spaces. That is, R-SDFAs will either accept or reject a trace with a given probability, while our models will generate a trace with a certain probability. For R-SDFAs, the following problems are undecidable: i) is there a trace that has a larger probability than a threshold, and ii) do there exist traces whose probability is arbitrarily close to a given probability [11,2]. It remains to be studied if such undecidability results carry over to our setting.

8 Conclusion

Stochastic process mining calls for answering probabilistic queries on stochastic process models. These are queries that retrieve traces satisfying a criterion expressed through conditions on probability masses. This is instrumental to to validate the model, to inspect the model if it is too complex to understand in full, to formulate predictions, and to check conformance.

In this paper, we have tackled the problem of answering such probability queries. We defined a stochastic deterministic state space of activities, and showed how it can be computed incrementally for unbounded SLPNs with livelocks and silent transitions, with the exception of unbounded silent cycles. On this state space, we showed how to answer three types of probability queries, and proved that answering such queries is computable if Conjecture 14 holds and two facts are known: whether the model has an unbounded number of traces, and the total probability that the model ends up in a livelock. The approach was implemented in the Ebi framework [17] and is open source. We evaluated its feasibility on real-life and artificial logs, and applicability on a real-life log.

As for future work, we plan to build on the connection between the q-state space as presented here and the stochastic automata models discussed in Section 7, towards establishing (un)decidability and complexity results related to the exact computation of probability queries for different SLPN classes. Furthermore, we suggest to further explore implementation improvements and strategies to truncate the resulting state space when answering probability queries. Finally, it would be interesting to explore an adaptation of our approach for stochastic Time Petri Nets, employing the results from [6] on transient probability calculations in semi-Markov processes.

Acknowledgments. Marco Montali has been partially funded by the NextGenerationEU FAIR PE0000013 project MAIPM (CUP C63C22000770006) and by the PRIN MIUR project PINPOINT Prot. 2020FNEB27.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. Blondel, V.D., Canterini, V.: Undecidable problems for probabilistic automata of fixed dimension. Theory Comput. Syst. **36**(3), 231–245 (2003)
3. Burke, A., Leemans, S.J.J., Wynn, M.T.: Discovering stochastic process models by reduction and abstraction. In: Petri Nets. LNCS, vol. 12734, pp. 312–336. Springer (2021)
4. Chiola, G., Marsan, M.A., Balbo, G., Conte, G.: Generalized stochastic Petri nets: A definition at the net level and its implications. IEEE Trans. on soft. eng. **19**(2), 89–107 (1993)
5. Cohen, I., Gal, A.: Uncertain process data with probabilistic knowledge: Problem characterization and challenges. In: PROBLEMS. vol. 2938. CEUR (2021)
6. Dengler, G., Carnevali, L., Budde, C.E., Vicario, E.: Transient evaluation of non-markovian models by stochastic state classes and simulation. In: QEST+FORMATS. Lecture Notes in Computer Science, vol. 14996, pp. 213–232. Springer (2024)
7. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. American Journal of Mathematics **35**(4), 413–422 (1913)
8. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
9. Esparza, J.: Petri nets lecture notes. PNSkript. pdf (2019)
10. Finkel, A.: The minimal coverability graph for petri nets. In: Applications and Theory of Petri Nets. Lecture Notes in Computer Science, vol. 674, pp. 210–243. Springer (1991)
11. Gimbert, H., Oualhadj, Y.: Probabilistic automata on finite words: Decidable and undecidable problems. In: ICALP. LNCS, vol. 6199. Springer (2010)
12. Grinstead, C.M., Snel, J.L.: Introduction to Probability, Second Revised Edition. American Mathematical Society (1997)
13. Hartmanns, A., Junges, S., Quatmann, T., Weininger, M.: A practitioner's guide to MDP model checking algorithms. In: TACAS (1). LNCS, vol. 13993. Springer (2023)
14. Kalenkova, A.A., Mitchell, L., Roughan, M.: Performance analysis: Discovering semi-markov models from event logs. CoRR **abs/2206.14415** (2022)
15. Leemans, S.J.J., van der Aalst, W.M.P., Brockhoff, T., Polyvyanyy, A.: Stochastic process mining: Earth movers' stochastic conformance. Inf. Syst. **102**, 101724 (2021)
16. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM Workshops. LNBIP, vol. 171. Springer (2013)
17. Leemans, S.J.J., Li, T., van Detten, J.N.: Ebi - a stochastic process mining framework. In: ICPM Doctoral Consortium / Demo. CEUR Workshop Proceedings, vol. 3783. CEUR-WS.org (2024)
18. Leemans, S.J.J., Li, T., Montali, M., Polyvyanyy, A.: Stochastic process discovery: Can it be done optimally? In: CAiSE. LNCS, vol. 14663. Springer (2024)
19. Leemans, S.J.J., Maggi, F.M., Montali, M.: Enjoy the silence: Analysis of stochastic petri nets with silent transitions. Inf. Sys. **to appear** (2024)
20. Leemans, S.J.J., Polyvyanyy, A.: Stochastic-aware precision and recall measures for conformance checking in process mining. Inf. Syst. **115**, 102197 (2023)

21. Leemans, S.J.J., Poppe, E., Wynn, M.T.: Directly follows-based process mining: Exploration & a case study. In: ICPM. pp. 25–32. IEEE (2019)
22. Leemans, S.J.J., Syring, A.F., van der Aalst, W.M.P.: Earth movers' stochastic conformance checking. In: BPM Forum. LNBIP (2019)
23. Li, T., Leemans, S.J., Polyvyanyy, A.: The jensen-shannon distance for stochastic conformance checking. In: International Conference on Process Mining. pp. 70–83. Springer (2024)
24. Marsan, M.A., Conte, G., Balbo, G.: A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. ACM Trans. Comput. Syst. **2**(2), 93–122 (1984). https://doi.org/10.1145/190.191, https://doi.org/10.1145/190.191
25. Puterman, M.L.: Markov decision processes. In: Stochastic Models, Handbooks in Operations Research and Management Science, vol. 2, pp. 331–434. Elsevier (1990)
26. Rabin, M.O.: Probabilistic automata. Inf. Control. **6**(3), 230–245 (1963)
27. Rogge-Solti, A., van der Aalst, W.M.P., Weske, M.: Discovering stochastic Petri nets with arbitrary delay distributions from event logs. In: BPM workshops. LNBIP, vol. 171. Springer (2013)
28. Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., Carrasco, R.C.: Probabilistic finite-state machines-part I. IEEE Trans. Pattern Anal. Mach. Intell. **27**(7), 1013–1025 (2005)
29. Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., Carrasco, R.C.: Probabilistic finite-state machines-part II. IEEE Trans. Pattern Anal. Mach. Intell. **27**(7), 1026–1039 (2005)