Lightweight Statistical Model Checking in Nondeterministic Continuous Time*

Pedro R. D'Argenio[1,2,3], Arnd Hartmanns[4], and Sean Sedwards[5]

[1] Universidad Nacional de Córdoba, Córdoba, Argentina
[2] CONICET, Córdoba, Argentina
[3] Saarland University, Saarbrücken, Germany
[4] University of Twente, Enschede, The Netherlands
[5] University of Waterloo, Waterloo, Canada

Abstract. Lightweight scheduler sampling brings statistical model checking to nondeterministic formalisms with undiscounted properties, in constant memory. Its direct application to continuous-time models is rendered ineffective by their dense concrete state spaces and the need to consider continuous input for optimal decisions. In this paper we describe the challenges and state of the art in applying lightweight scheduler sampling to three continuous-time formalisms: After a review of recent work on exploiting discrete abstractions for probabilistic timed automata, we discuss scheduler sampling for Markov automata and apply it on two case studies. We provide further insights into the tradeoffs between scheduler classes for stochastic automata. Throughout, we present extended experiments and new visualisations of the distribution of schedulers.

1 Introduction

Statistical model checking (SMC [24,33]) is a formal verification technique for stochastic systems based on Monte Carlo simulation. It naturally works with non-Markovian behaviour and complex continuous dynamics that make the exact model checking problem intractable. As a simulation-based approach, however, SMC is incompatible with nondeterminism. Yet (continuous and discrete) nondeterministic choices are desirable in formal modelling, for abstraction and to represent concurrency as well as the absence of knowledge. Nondeterminism occurs in many popular formalisms, notably in Markov decision processes (MDP). In the presence of nondeterminism, quantities of interest are defined w.r.t. optimal *schedulers* (also called policies, adversaries or strategies) resolving all nondeterministic choices: the verification result is the *maximum* or *minimum* probability or expected value ranging over *all* schedulers. Many SMC tools appear to support nondeterministic models, e.g. Prism [28] and Uppaal smc [13], but use a single implicit probabilistic scheduler that makes all choices randomly. Their

* This work is supported by the 3TU project "Big Software on the Run", by ERC grant 695614 (powver), by the JST ERATO HASUO Metamathematics for Systems Design project (JPMJER1603), and by SeCyT-UNC projects 05/BP12, 05/B497.

results thus lie *somewhere* between the minimum and maximum. Such implicit resolutions are known to affect the trustworthiness of simulation studies [3,27].

Sound SMC in the presence of nondeterminism is a hard problem. For MDP, Brázdil et al. [4] proposed a sound machine learning technique, while Uppaal Stratego [12] explicitly synthesises a "good" scheduler before using it for a standard SMC analysis. Both approaches suffer from worst-case memory usage linear in the number of states. Classic memory-efficient sampling approaches (e.g. [25]) address discounted models only. In contrast, the modes tool [6], part of the Modest Toolset [20], extends the lightweight scheduler sampling (LSS) approach for MDP first implemented in Plasma [30]. LSS is the only technique that applies to undiscounted properties, as typically considered in formal verification, that also keeps memory usage effectively constant in the number of states.

The effectiveness of LSS depends on the probability of sampling near-optimal schedulers. It works well for discrete-time discrete-state models like MDP, where memoryless schedulers achieve optimal probabilities on a discrete state space. Yet the concrete state spaces of continuous-time models may be uncountably infinite, and optimal schedulers may need real-valued input based on model time. This renders naive applications of scheduler sampling ineffective. However, the use of suitable discrete abstractions makes the approach both feasible and useful for some continuous-time formalisms.

This paper summarises, connects and extends previous work on LSS for continuous-time models. After an introduction to the concept of LSS on MDP in Sect. 2, we summarise recent extensions to probabilistic timed automata (PTA [29]) using regions [22] and zones [9] in Sect. 3. We report extended experimental results, sampling more schedulers and reducing the statistical error compared to our previous work. In Sect. 4 we investigate the challenges in extending LSS to Markov automata (MA [14]), a compositional nondeterministic extension of continuous-time Markov chains. We introduce two new case studies to experiment with modes' support for LSS on MA. In Sect. 5 we turn to stochastic automata (SA [10]), which include general continuous probability distributions. We have recently shown that no simple class of schedulers achieves optimal probabilities on SA [8]. We summarise these results and their effect on LSS, and provide more detailed experimental results to investigate the tradeoffs between restricted classes and discrete abstractions of the state space.

All methods described in this paper are implemented in the modes statistical model checker [6], which was used to perform all the experiments. To investigate the distribution of schedulers, we extended modes to create histograms that visualise the distribution of schedulers w.r.t. the probabilities they induce. We present histograms for all our experiments, providing deeper insights into the character of the nondeterminism in the models and the behaviour of LSS.

2 Lightweight Statistical Model Checking

We summarise the lightweight scheduler sampling approach for Markov decision processes [30], which is the foundation of our techniques for timed systems.

Definition 1. *A (discrete)* probability distribution *over a set Ω is a function $\mu \in \Omega \to [0,1]$ such that* $\mathrm{support}(\mu) \stackrel{\text{def}}{=} \{\, \omega \in \Omega \mid \mu(\omega) > 0 \,\}$ *is countable and* $\sum_{\omega \in \mathrm{support}(\mu)} \mu(\omega) = 1$. $\mathrm{Dist}(\Omega)$ *is the set of all probability distributions over Ω.*

Definition 2. *A* pseudo-random number generator *(PRNG) $\mathcal{U}$ can be initialised with a* seed *$i \in \mathbb{N}$ ($\mathcal{U} := \mathrm{PRNG}(i)$) and then* iterated *($\mathcal{U}()$) to produce a new value pseudo-uniformly distributed in $[0,1)$ and pseudo-statistically independent of previous iterates. For a given i, the sequence of iterates is always the same. We denote by $\mathcal{U}(\mu)$ the pseudo-random selection of a value from* $\mathrm{support}(\mu)$ *according to a value sampled from $\mathcal{U}$ and the probabilities in $\mu \in \mathrm{Dist}(\Omega)$.*

Markov decision processes combine nondeterminism and probabilistic choices. To move from one state to another, first a transition is chosen nondeterministically. Every transition leads into a probability distribution over successor states.

Definition 3. *A* Markov decision process *(MDP) is a triple $M = \langle S, T, s_{init} \rangle$ where S is a countable set of states, $T \in S \to 2^{\mathrm{Dist}(S)}$ is the transition function with $T(s)$ countable for all $s \in S$, and $s_{init} \in S$ is the initial state. If $|T(s)| \leq 1$ for all $s \in S$, then M is a* discrete-time Markov chain *(DTMC).*

A *transition* is a pair $\langle s, \mu \rangle$ s.t. $\mu \in T(s)$. A *path* in an MDP is an infinite sequence $\langle s_0, \mu_0 \rangle \langle s_1, \mu_1 \rangle \ldots$ of transitions with $s_0 = s_{init}$. When the current state is s_i, the nondeterministic choice of the next transition is made by a scheduler:

Definition 4. *A (memoryless deterministic)* scheduler *for an MDP is a function $\mathfrak{s} \in S \to \mathrm{Dist}(S)$ s.t. $\mathfrak{s}(s) \in T(s)$ for all $s \in S$. $\mathfrak{S}$ is the set of all schedulers.*

Once a transition $\langle s_i, \mu_i \rangle$ is chosen, the next state s_{i+1} is selected randomly according to μ_i. Restricting to the choices made by $\mathfrak{s}$ induces a DTMC, and $\mathfrak{s}$ defines the probability measure $\mathbb{P}_{\mathfrak{s}}$ over paths [16]. *Transient properties* ϕ are queries for the optimal probabilities $\mathrm{opt}_{\mathfrak{s} \in \mathfrak{S}} \mathbb{P}_{\mathfrak{s}}(\neg \mathit{avoid}\ \mathtt{U}\ \mathit{target})$ where $\mathrm{opt} \in \{\, \sup, \inf \,\}$ (for *maximum* and *minimum* probabilities, denoted $p_{\max}$ and $p_{\min}$), $\mathit{avoid}, \mathit{target} \subseteq S$, and $\neg \mathit{avoid}\ \mathtt{U}\ \mathit{target}$ is the set of paths with at least one state in *target* such that no state in *avoid* has been visited earlier. For these properties, the restriction to memoryless deterministic schedulers preserves optimal probabilities. For a finite *trace* ω, i.e. a path prefix projected to its states, let $\phi(\omega)$ be *undecided* if ω does not contain a state in $\neg \mathit{avoid} \cup \mathit{target}$, *true* if ϕ is satisfied on all paths that have a prefix projecting to ω, and *false* otherwise.

Using MDP to directly model complex systems is cumbersome. Instead, higher-level formalisms like Modest [18] are used. They provide parallel composition and finite-domain variables. This allows to compactly describe very large MDP. Modest in fact supports all of the formalisms introduced in this paper.

Statistical model checking (SMC) [24,33] is, in essence, Monte Carlo integration of formal models. It generates a large number n of simulation *runs* according to the probability distributions in the model and uses them to statistically estimate the probability for a given property. For transient property ϕ on a DTMC,

Input: MDP $M = \langle S, T, s_{init} \rangle$, transient property ϕ, scheduler identifier $\sigma \in \mathbb{Z}_{32}$
Output: Sampled trace ω

1 $s := s_{init}$, $\omega := s_{init}$
2 **while** $\phi(\omega) = \mathit{undecided} \wedge T(s) \neq \varnothing$ **do** // *run until ϕ decided or deadlock*
3 $\quad \mathcal{U}_{\text{nd}} := \text{PRNG}(\mathcal{H}(\sigma.s))$ // *seed $\mathcal{U}_{\text{nd}}$ with hash of σ and s*
4 $\quad \mu := \lceil \mathcal{U}_{\text{nd}}() \cdot |T(s)| \rceil$-th element of $T(s)$ // *use $\mathcal{U}_{\text{nd}}$ to select a transition*
5 $\quad s := \mathcal{U}_{\text{pr}}(\mu)$, $\omega := \omega.s$ // *use $\mathcal{U}_{\text{pr}}$ to select next state, append to ω*
6 **return** ω

Algorithm 1. Lightweight simulation for an MDP and a scheduler identifier

the runs are traces $\omega_1, \ldots, \omega_n$ such that $\phi(\omega_i) \neq \mathit{undecided}$, and the estimate is $\hat{p}_n = \frac{1}{n} \sum_{i=0}^{n} \phi(\omega_i)$ when identifying *true* with 1 and *false* with 0. $\hat{p}_n$ is an unbiased estimator of the actual probability p. The choice of n depends on the desired statistical properties of $\hat{p}$, e.g. that a confidence interval around $\hat{p}$ with confidence δ has half-width w. For a detailed description of statistical methods and especially hypothesis tests for SMC, we refer the reader to [32].

Lightweight scheduler sampling (LSS) extends SMC to the nondeterministic model of MDP by approximating optimal schedulers, i.e. those that realise $p_{\min}$ or $p_{\max}$, in constant memory relative to the size of the state space [30]. A scheduler is identified by a single (32-bit) integer. LSS randomly selects m schedulers (i.e. integers), performs standard SMC on the DTMC induced by each, and reports the maximum and minimum estimates over all sampled schedulers as approximations of the actual respective probabilities. We show the core of LSS—performing a simulation run for a given scheduler identifier σ—as Alg. 1. It uses two PRNGs: $\mathcal{U}_{\text{pr}}$ is initialised globally once and used to simulate the probabilistic choices of the MDP in line 5, while $\mathcal{U}_{\text{nd}}$ resolves the nondeterministic choices in line 4. We want σ to represent a deterministic memoryless scheduler. Therefore, within one simulation run as well as in different runs for the same value of σ, $\mathcal{U}_{\text{nd}}$ must always make the same choice for the same state s. To achieve this, $\mathcal{U}_{\text{nd}}$ is re-initialised with a seed based on σ and s in every step (line 3).

The effectiveness of LSS depends on the probability of sampling a near-optimal scheduler. Since we do not know a priori what makes a scheduler optimal, we want to sample "uniformly" from the space of all schedulers. This at least avoids actively biasing against "good" schedulers. More precisely, a uniformly random choice of σ will result in a uniformly chosen (but fixed) resolution of all nondeterministic choices. Alg. 1 achieves this naturally for MDP.

Bounds and error accumulation. The results of LSS are lower bounds for maximum and upper bounds for minimum probabilities up to the specified statistical error. They can thus be used to e.g. *disprove* safety or *prove* schedulability, but not the opposite. The accumulation of statistical error introduced by the repeated simulation experiments over m schedulers must also be accounted for, using e.g. Šidák correction or the modified tests described in [11].

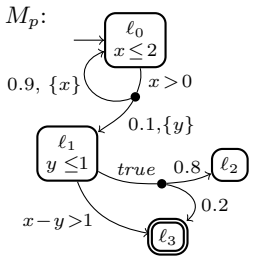

Fig. 1. Example PTA M_p

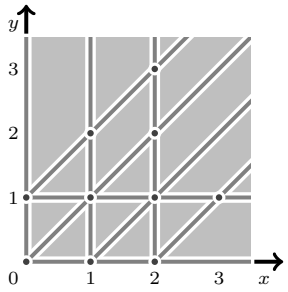

Fig. 2. Regions of M_p

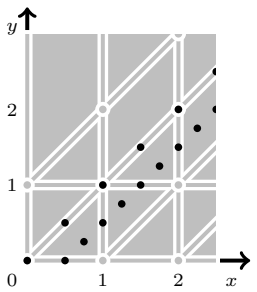

Fig. 3. Representatives

Two-phase and smart sampling. If, for fixed statistical parameters, SMC needs n runs on a DTMC, LSS needs significantly more than $m \cdot n$ runs on an MDP to avoid error accumulation. The *two-phase* and *smart* sampling approaches can reduce this overhead. The former's first phase consists of performing n simulation runs for each of the m schedulers. The scheduler that resulted in the maximum (or minimum) value is selected, and independently evaluated once more with n runs to produce the final estimate. The first phase is a heuristic to find a near-optimal scheduler before the second phase estimates the value under this scheduler according to the required statistical parameters. Smart sampling [11] generalises this principle to multiple phases, dropping the "worst" half of the schedulers in every round. It tends to find better schedulers faster, while the two-phase approach has predictable performance: it always needs $(m + 1) \cdot n$ runs. We use the two-phase approach for all experiments reported in this paper.

3 Probabilistic Timed Automata

Probabilistic timed automata (PTA [29]) combine MDP and timed automata [1]. We show an example PTA M_p in Fig. 1. It has two *clocks* x and y: variables over $[0, \infty)$ that advance synchronously with rate 1 as time passes. As PTA are a symbolic model, we speak of *locations* (in *Loc*) and *edges* instead of states and transitions. M_p has locations ℓ_0 through ℓ_3. Every location is associated with a *time progress condition*: $x \leq 2$ in ℓ_0, $y \leq 1$ in ℓ_1, and *true* elsewhere. These are *clock constraints*: expressions of the form $\mathcal{C} ::= true \mid false \mid \mathcal{C} \wedge \mathcal{C} \mid c \sim n \mid c_1 - c_2 \sim n$ where $\sim \in \{>, \geq, <, \leq\}$, c, c_1, c_2 are clocks, and $n \in \mathbb{N}$. Every edge is annotated with a *guard* clock constraint and sets of clocks to *reset* to zero. M_p has one edge out of ℓ_0 with guard $x > 0$ that goes back to ℓ_0 with probability 0.9, resetting x, and otherwise to ℓ_1, resetting y. There are two edges out of ℓ_1. The one with guard $x - y > 1$ goes to ℓ_3 with probability 1 and no resets.

Intuitively, the semantics of a PTA is an uncountably infinite MDP: Its states are pairs $\langle \ell, v \rangle$ of the current location ℓ and valuation v for all clocks. In ℓ, time can pass (i.e. the values in v increase) as long as the time progress condition remains satisfied. An edge can be taken if its guard evaluates to *true* at the current point in time. Then a target is chosen randomly, the specified clocks are

reset to zero, and we move to the target location. Writing valuations as tuples $\langle v(x), v(y)\rangle$, one concrete trace in the semantics of M_p is

$$\langle \ell_0, \langle 0,0\rangle\rangle \langle \ell_0, \langle 0.8, 0.8\rangle\rangle \langle \ell_0, \langle 0, 0.8\rangle\rangle \langle \ell_0, \langle 1.1, 1.9\rangle\rangle \langle \ell_1, \langle 1.1, 0\rangle\rangle \langle \ell_3, \langle 1.1, 0\rangle\rangle.$$

The time spent in ℓ_0 and ℓ_1 is nondeterministic, as is the choice of edge in ℓ_1.

The transient properties defined for MDP in Sect. 2 apply analogously to PTA. In addition, time-bounded properties—where *target* must be reached in $\leq d \in \mathbb{N}$ time units—can be encoded as unbounded ones by adding a new clock c_d that is never reset and replacing *target* by $\{\, \langle \ell, v\rangle \mid \ell \in \mathit{Loc} \wedge v(c_d) \leq d \,\} \cap \mathit{target}$. In M_p, the minimum probability to reach l_3 is 0.2. The maximum is 1; it is only achieved by always waiting in l_0 until $x > 1$ before taking the edge.

A naive extension of lightweight SMC to PTA is to use Alg. 1 to generate concrete traces like the one given for M_p above. The input to $\mathcal{U}_{\text{nd}}$ is then a hash of σ and the current state $\langle \ell, v\rangle$. $\mathcal{U}_{\text{nd}}$ selects a delay in $[0, \infty)$ permitted by the time progress condition, followed by an enabled edge, if available. However, this can make (near-)optimal schedulers infeasibly rare. Consider M_p and the maximum probability to reach ℓ_3. An optimal scheduler must *always* select a delay > 1 in ℓ_0. Yet, for a fixed σ, we get to make a *new* decision every time we come back to ℓ_0 because $v(y)$ most likely is a different real number in $[0, 2]$ every time. The probability of choosing a σ that *always* makes the *same* decision is zero, and even near-optimal schedulers are rare. The problem is that the number of critical decisions is infinite, such that optimal schedulers have measure zero. To be effective, LSS needs the number of critical decisions to be finite.

3.1 Lightweight SMC with Discrete Abstractions

To model-check transient properties, it suffices to consider the finite *region* graph of a PTA [29], a concept first introduced for timed automata [1]. Since it is too large to be useful in practice, timed automata verification tools instead use *zones*.

Definition 5. *Let $k_c \in \mathbb{N}$ be the maximum constant appearing in comparisons with clock c. A* zone *is a non-empty set of valuations that can be described by a clock constraint in which all comparisons have the form $c_1 - c_2 \sim n_{c_1c_2}$ for $n_{c_1c_2} \in \{\, 0, \ldots, \max\{\, k_{c_1}, k_{c_2}\,\}\,\}$ or $c \sim n_c$ for $n_c \in \{\, 0, \ldots, k_c\,\}$. A* region *$r$ is a minimal zone; its* successor *is the unique first other region encountered when delaying from any valuation in r.*

In M_p we have $k_x = 2$ and $k_y = 1$. The regions of M_p are visualised in Fig. 2: Every gray point, line segment and area is a region. To find a region's successor, follow a 45-degree line from any point within the region up to the next region.

We could use Alg. 1 on the region graph. However, if the only available operations on regions are to (1) reset a clock and (2) obtain the successor, then performing a long delay needs many simulation steps to sequentially move through several successors. This causes significant performance problems and prevents uniform scheduler sampling: As long as the time progress condition is

satisfied, the only reasonable way to implement the scheduler is to let $\mathcal{U}_{\text{nd}}$ choose uniformly between delaying to the successor or taking an edge. The total delay thus follows a geometric distribution, biasing towards taking edges early.

A zone-based approach [9] using the standard difference-bound matrix (DBM) data structure solves these two problems. We can easily obtain and represent an entire sequence of regions as a single zone, determine the edges enabled throughout that zone, and use $\mathcal{U}_{\text{nd}}$ to uniformly (but deterministically for fixed σ) select one. The resulting algorithm (shown as Alg. 2 in [9]) is not a simple extension of Alg. 1 for several reasons that we explore in that paper. In particular, when taking an edge, it needs to select a single region from within the target zone. This is to avoid over-/underapproximating probabilities, since it performs a forwards exploration [29]. The drawback of zone-based LSS is performance: The runtime of most DBM operations, such as intersecting two zones or resetting clocks, is cubic in the number of clocks [2], and selecting a region uniformly at random is exponential [9]. We use a faster quasi-uniform algorithm in our experiments.

Efficient simulation with regions became possible with our new efficient data structure for regions that supports long delays without enumerating successor regions [22]. It implements all operations with worst-case runtime linear in the number of clocks. The problem of efficient data structures for regions had received scant attention as the region graph is too large for exact model checking.

A straightforward symbolic representation of regions consists of a mapping from each clock to the integer part of its value, plus a total order of the fractional parts. Our data structure additionally provides a concrete *representative* value in $\mathbb{Q}$ for every clock, and a function that, given a delay based on a representative valuation, performs that entire delay in one go. The concrete choice of representatives is the main insight. For every clock, the representative value is a multiple of $1/(2 \cdot n_d)$, where n_d is the number of different fractional values among all clocks. We show the representatives of regions of M_p as black dots in Fig. 3: the one of region $x = y = 0$ (which has $n_d = 1$), the one of $0 < x < y \wedge y = 0$ (with $n_d = 2$), their successors, and so on. This choice of representatives is the only one where representatives are equally spaced, allowing an efficient implementation of the delay function. The resulting LSS core is shown as Alg. 3 in [22].

3.2 Experiments

In [22] we compared the zone- and region-based approaches on PTA models of communication protocols from the literature. We estimated the probabilities of
- termination in 4000 ns in IEEE 1394 FireWire root contention (*firewire*),
- either of two stations' backoff counters reaching value 2 within one transmission in IEEE 802.11 wireless LAN (*wlan*) using the original timing parameters from the standard (e.g. a maximum transmission time of 15717 µs), and
- all stations correctly delivering their packets within D_n µs on a shared medium via the exponential backoff procedure in IEEE 802.3 CSMA/CD with $n \in \{2, 3, 4\}$ stations (*csmacd*$_n$), using $D_2 = 1800$, $D_3 = 2700$ and $D_4 = 3600$.

Table 1. Performance and results for PTA

		model checking		ad-hoc schedulers			LSS with regions			LSS with zones		
model	clocks	$p_{\min}$	$p_{\max}$	ALAP	Uniform	ASAP	time	$\hat{p}_{\min}$	$\hat{p}_{\max}$	time	$\hat{p}_{\min}$	$\hat{p}_{\max}$
firewire	1+1	0.781	1.000	0.95	0.98	1.00	20 s	0.79	1.00	27 s	0.79	1.00
wlan	2		0.063	0.05	0.05	0.05	2 744 s	0.04	0.06	3 903 s	0.04	0.06
$csmacd_2$	4+1	0.729	0.872	0.73	0.75	0.87	108 s	0.73	0.85	398 s	0.73	0.87
$csmacd_3$	5+1	0.663	0.892	0.71	0.81	0.89	312 s	0.78	0.85	1 185 s	0.77	0.87
$csmacd_4$	6+1			0.68	0.83	0.90	656 s	0.80	0.85	2 555 s	0.80	0.86

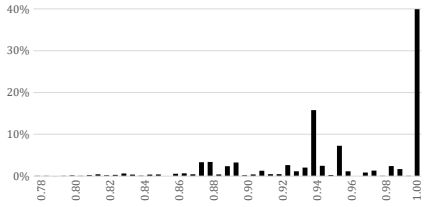

Fig. 4. Histogram for *firewire* (regions)

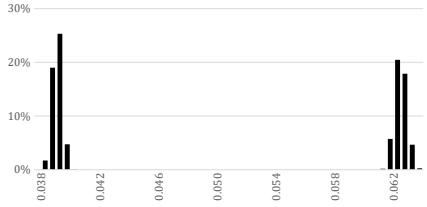

Fig. 5. Histogram for *wlan* (regions)

In Table 1 we report the results of a new set of experiments on these models and properties. We have modified the zone-based approach to greedily try to enter/avoid the *target* and *avoid* sets for maximum probabilities (and vice-versa for minimum probabilities) after identifying the set of delays allowed by the time progress condition but before selecting an edge. We also improved the fast quasi-uniform region selection algorithm. Furthermore, we sample more schedulers ($m = 1000$) and have reduced the statistical error: We use $n = 372221$ runs per scheduler for *wlan* and 14889 for the other models. Via the Okamoto bound [31] (as used in the "APMC method" [24]), which relates n to values ϵ and δ s.t. $\mathbb{P}(|\hat{p} - p| > \epsilon) < \delta$ for estimate $\hat{p}$ and actual probability p, this guarantees $\epsilon = 0.001$ for *wlan* and $\epsilon = 0.005$ for the other models with confidence $\delta = 0.95$. We also compare with the probabilities induced by three ad-hoc schedulers:

- **Uniform** selects uniformly at random among the time points where ≥ 1 edge is enabled before uniformly selecting one edge enabled after that chosen delay;
- **ASAP** instead selects the first time point where any edge is enabled; and
- **ALAP** always picks the last time point where at least one edge is enabled.

These are *randomised* schedulers: they may make a different choice every time the same state is visited. They also require the intersections of guards and time progress conditions to be bounded, which is the case for all three models. The Uniform scheduler is similar to the implicit one of Uppaal smc [13]. All experiments were performed on a cluster of 10 Intel Xeon E5520 nodes (2.26-2.53 GHz) with 64-bit Ubuntu Linux, providing 40 simulation threads in total (4 per node). Every experiment was performed three times and we report the averages.

Discussion. As expected and previously shown in [22], the region-based approach significantly outperforms the zone-based one as the number of clocks grows. On the larger *csmacd* models, however, the latter finds better schedulers. Comparing with the ad-hoc schedulers reveals that long (short) delays lead to worse (better)

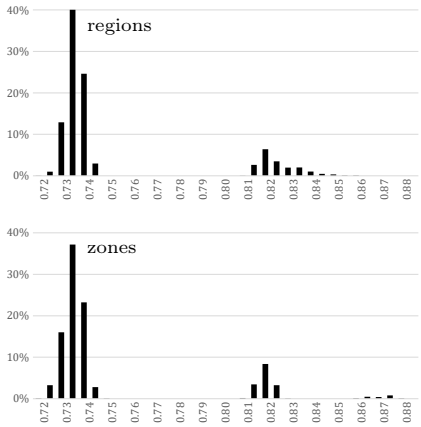

Fig. 6. Histograms for $csmacd_2$

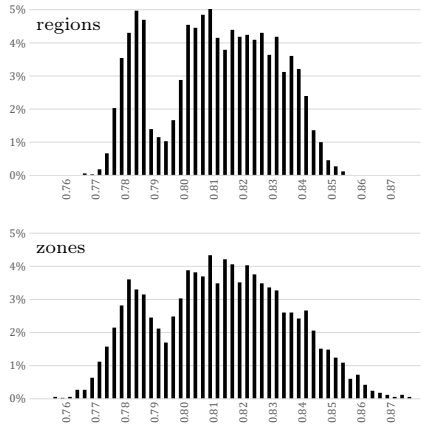

Fig. 7. Histograms for $csmacd_3$

performance of the protocol, with the results of the ALAP and ASAP schedulers being closer to the actual optimal probabilities (which we could exactly model-check for the smaller models) than any scheduler found via LSS. So if always scheduling fast or slow indeed is optimal, then near-optimal schedulers are rare: they must *always* pick the min. or max. delay, with the delay choices increasing as the number of stations grows. On *firewire*, ad-hoc schedulers only lead to probabilities near the maximum, while LSS also finds near-minimal schedulers.

Scheduler histograms. We extended modes to also return the probabilities estimated for *all* m sampled schedulers. This allows us to create histograms that visualise the distribution of schedulers w.r.t. the probabilities they induce. The histograms for *firewire* and *wlan* using regions are shown in Figs. 4 and 5, respectively, with the ones for zones being nearly identical. We see the reasons for the success of LSS as well as the failure of the ad-hoc schedulers reflected in these histograms: For *firewire*, maximal schedulers are very likely while minimal ones are rarer, but still show decent probabilities. For *wlan*, every *deterministic* scheduler sampled by LSS is either near-minimal or near-maximal; the *randomised* ad-hoc schedulers however only realise an average of these two behavioural modes.

For *csmacd*, the distributions of schedulers found with regions and zones are clearly different. With two stations (Fig. 6), there are distinct clusters of similar schedulers, however the region-based approach does not find good ones in the near-maximal cluster. As the number of stations and thus of nondeterministic decisions increases, the average sampled scheduler leads to more average behaviour (Fig. 7), yet the variance among zone-based schedulers is still wider.

4 Markov Automata

Markov automata (MA, [14]) are a compositional model that combines the discrete probabilistic branching of MDP with the exponentially distributed delays of

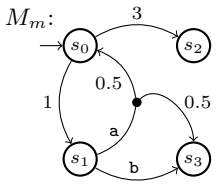

Fig. 8. Example MA M_m

Table 2. Performance and results for MA

model		m	n	$\lvert\omega\rvert$	time	$\hat{v}_{\min}$	$\hat{v}_{\max}$
queues	Unif.	1	372 k	25	1 s	0.096	
	LSS	100	372 k	25	91 s	0.043	0.144
		1 000	372 k	25	872 s	0.031	0.170
bitcoin	Unif.	1	433	8 k	2 s	26 701	
	LSS	1 000	456	14 k	332 s	6 926	249 323
		10 000	433	13 k	2 900 s	6 561	233 745

continuous-time Markov chains (CTMC). We show an example MA with states s_0 through s_3 in Fig. 8. It has two types of transitions: *Markovian* ones (as in CTMC) labelled with a rate in $(0, \infty)$ connect s_0 to s_1 and s_2, while *probabilistic* transitions (as in MDP) connect s_1 to s_3 and back to s_0. The *exit rate* of s_0 is $1 + 3 = 4$. Probabilistic transitions are taken immediately when available, with the choice between multiple transitions (like `a` and `b` in s_1) being nondeterministic. Markovian transitions become enabled after an amount of time has passed that is exponentially distributed according to the rate of the transition. The choice between multiple of them is resolved by a race between the distributions.

In terms of properties, we are interested in unbounded and time-bounded transient properties, as for PTA. However, due to the absence of clocks, time-bounded properties cannot be encoded as unbounded ones. They instead need to be supported by dedicated analysis methods. We also use expected-time properties to calculate the minimum and maximum expected times $t_{\min}$ and $t_{\max}$ until a set of *target* states is reached for the first time. We require probability 1 for *true* `U` *target*. For transient property *true* `U` $\{ s_3 \}$ in M_m, we have $p_{\min} = 0.6$ (always schedule `a`) and $p_{\max} = 0.75$ (always schedule `b`). For the expected time to reach $\{ s_2, s_3 \}$, we have $t_{\max} = 0.4$ and $t_{\min} = 0.25$ with the same schedulers.

4.1 Lightweight SMC Possibilities and Challenges

The application of LSS to MA with unbounded transient and expected-time properties is a straightforward adaption to MA of Alg. 1, since memoryless deterministic schedulers are sufficient to obtain optimal results [17,23].

For time-bounded properties, optimal schedulers need to take into account the amount of time remaining until the time bound is reached. A naive extension of LSS would thus face the same issues as with PTA. The current approaches to perform exact model checking of a time-bounded property with bound d are to use either *digitisation* [23] or *uniformisation* [7]. The former discretises the MA by assuming that ≤ 1 Markovian transitions will fire within any small time interval $(0, \delta]$, where $\delta > 0$ is the digitisation constant such that $\exists\, k_b \in \mathbb{N}\colon d = k_b \cdot \delta$. Every state of the digitised model is a pair of the original state in the MA and the amount of time—a multiple of δ—remaining until d. That is, the model is unfolded over the time bound. If the maximum exit rate λ in the MA is known, then we also know that the max. probability computed on the digitised model is at most $k_b \cdot \frac{(\lambda\delta)^2}{2}$ below the actual one. As the digitised model is discrete, a

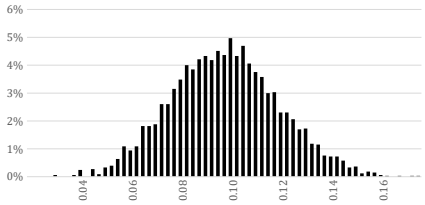

Fig. 9. Histogram for *queues*

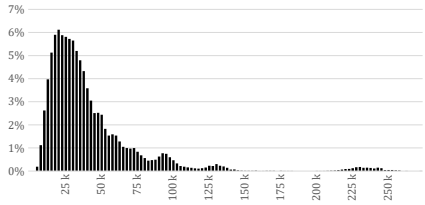

Fig. 10. Histogram for *bitcoin*

variant of Alg. 1 could be applied to it directly. However, for the error to be small, a fine digitisation is needed. For example, to achieve error ≤ 0.01 for $d = 0.5$ on M_m requires $\delta = 0.0025$ and $k_b = 200$. That is, the model is unfolded 200 times, so schedulers face the nondeterministic choice between `a` and `b` up to 200 times. The probability of sampling an optimal scheduler (i.e. one that always makes the optimal choice) is then 0.5^{200}. Uniformisation, on the other hand, requires global information—the maximum exit rate λ, or an overapproximation thereof—to be applicable in the first place. Furthermore, it does not provide an a priori error bound. When used for model checking, the error is bounded by simultaneously computing an over- and underapproximation of the (max.) probability. However, LSS intrinsically underapproximates and introduces a statistical error. Finally, it is currently not clear how to efficiently apply the method of [7] in an on-the-fly manner as required for simulation. Further research into methods for effective LSS with time-bounded properties on MA is thus needed.

4.2 Experiments

We have implemented LSS for unbounded properties on MA in modes [6]. We evaluate the implementation on two new case studies with properties based on non-rare events (rather than the rare-event *database* model of [6]). We consider

- the queueing system with breakdowns (*queues*) of [26] where ten sources of two types produce packets and fail at different rates. A single server processes the packets and may also fail. We studied a deterministic version of this model in [5]. To experiment with LSS, we now model a single repairman that repairs one broken component at a time instead. If multiple components are broken, the next one to repair is selected nondeterministically. We estimate the probability for $\neg\, \mathit{reset}\ \mathtt{U}\ \mathit{buf} = 8$: starting from a single broken source, what is the probability for server queue overflow before all components are repaired?
- a Modest MA variant of the model of the *Andresen attack on Bitcoin* presented in [15] where a malicious pool of miners attempts to fork the blockchain to allow e.g. double spending. The malicious pool's strategy is kept open as nondeterministic choices in our model (*bitcoin*), and we estimate the expected time in minutes until the malicious pool succeeds at 20 % hash rate.

The experimental setup is as described in Sect. 3.2. All results are shown in Table 2, again including the Uniform ad-hoc scheduler for comparison. v stands for probabilities p for *queues* and for expected times t for *bitcoin*.

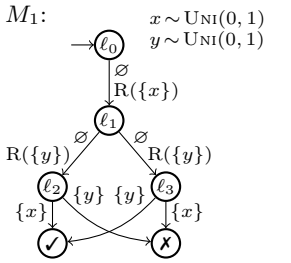

Fig. 11. SA M_1

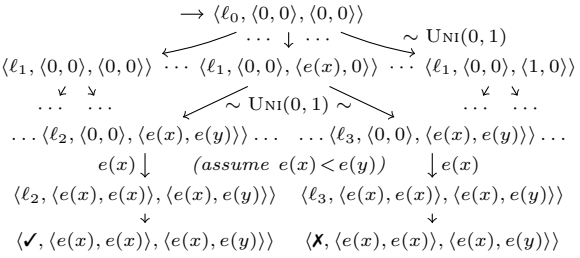

Fig. 12. Excerpt of the semantics of M_1

Discussion. To judge the rarity of near-optimal schedulers, we perform two LSS runs where the second samples 10 times as many schedulers (column m). For *queues*, sampling more schedulers improves the estimates: extremal schedulers are neither frequent nor excessively rare. This is confirmed by the histogram shown in Fig. 9. The Uniform scheduler again only obtains some average behaviour. When it comes to the *bitcoin* model, the histogram in Fig. 10 shows that the most frequently sampled schedulers achieve low expected times, i.e. they correspond to good strategies for the malicious pool. However, for the "default" and "optimised" strategies of [15], the expected times are 5403 and 3582 minutes. It is clear from the results in Table 2 that the sampled schedulers only come somewhat close to the default strategy in *absolute* terms. *Relative* to the worst schedulers found, however, they are still close to both good strategies. Once more, the Uniform scheduler is mostly useless here. In terms of performance, simulations for *bitcoin* take relatively long, which is due to the many simulation steps per run (column $|\omega|$) until the malicious pool wins with a bad strategy.

5 Stochastic Automata

Stochastic automata (SA, [10]) go beyond MA by (1) allowing delays to follow arbitrary probability distributions and (2) lifting the MA restriction of nondeterminism to (immediate) probabilistic edges. We show an example SA M_1 with six locations in Fig. 11. It has *stochastic clocks* x and y. The *expiration times* $e(x)$ and $e(y)$ follow the continuous uniform distribution over the interval $[0,1]$. An edge in an SA is guarded by a set of clocks: the edge becomes enabled and time cannot pass further as soon as all clocks in the *guard set* are expired. Thus no time can pass in ℓ_0 and ℓ_1. When taking an edge, clocks can be *restarted*: their values are reset to zero and their expiration times are resampled. On entering ℓ_1, x is restarted: its value $v(x)$ becomes zero and $e(x)$ is set to a random value selected from $\text{Uni}(0,1)$. The choice of going to either ℓ_2 or ℓ_3 from ℓ_1 is nondeterministic, since both outgoing edges become enabled simultaneously. Then y is restarted. In ℓ_2, we have to wait until the first of the two clocks expires. If that is x, we have to move to location ✓; if it is y, we have to move to ✗. The semantics of an SA is an uncountably infinite MDP similar to the semantics of a PTA,

but additionally with continuous distributions. The states in the semantics of M_1 are tuples of the form $\langle \ell, \langle v(x), v(y) \rangle, \langle e(x), e(y) \rangle \rangle$: they comprise the current location, the values of the clocks, and their expiration times. Nondeterministic choices are finite since they are between edges only. We illustrate a part of the semantics of M_1 as intuitively explained above in Fig. 12.

5.1 The Power of Schedulers for SA

We consider unbounded transient properties only. On M_1, the maximum probability for $\mathit{true}\ \mathtt{U}\ \{\checkmark\}$ is 0.75. It is achieved by going from ℓ_1 to ℓ_2 iff $e(x) \leq 0.5$: although the scheduler does not know in ℓ_1 what the expiration time of y is going to be after the restart of y, it is more likely to be higher than the (known) expiration time of x if that is low. This example shows that, in order to schedule optimally on SA, schedulers need to know the expiration times. We investigated the power of various restricted classes of schedulers for SA [8] and found that, aside from the history of previously visited states and delays, *all* components of the states are relevant for optimal scheduling. Let us write $\mathfrak{S}^a_{\ell,b,c}$ with $a \in \{\mathit{hist}, \mathit{ml}\}$, $b \in \{v, t, \text{-}\}$ and $c \in \{e, o, \text{-}\}$ to refer to a class of schedulers. Class $\mathfrak{S}^{\mathit{hist}}_{\ell,v,e}$ is the most general one: it sees the entire history (hist), clock values (v), and expiration times (e). We considered the following restrictions:

- memoryless schedulers that only see the current state (ml instead of hist),
- global-time schedulers that only see the total time elapsed since the initial state instead of the values of all individual clocks (t instead of v),
- schedulers that see the relative expiration order, i.e. the order of $e(z) - v(z)$ over all clocks z, in place of the expiration times (o instead of e), and
- schedulers that do not see some of the information at all (indicated by -).

Our findings include that all history-dependent schedulers seeing e coincide with class $\mathfrak{S}^{\mathit{ml}}_{\ell,v,e}$, and that for memoryless schedulers, knowing the expiration order o is incomparable to knowing e but not all of v. Where scheduler classes are not equivalent, we provided small distinguishing SA similar to M_1, which itself distinguishes all pairs of classes that only differ in seeing either e or o. We refer the interested reader to [8] (open-access) for a complete list of these six SA.

5.2 Lightweight SMC Possibilities and Challenges

Clearly, the naive extension of Alg. 1 to SA fails for the same reasons as for PTA. However, as we explained above and in contrast to MA, not even unbounded properties can be analysed via LSS by relying on a discrete class of schedulers. At the same time, many of the considered classes of schedulers are unrealistically powerful to consider as adversaries in a safety model checking scenario, and need too much information to be useful for implementation as strategies in a planning setup. For example, in many models the expiration times represent information about future events, thus using e or o leads to *prophetic* schedulers [21]. LSS based on some of the restricted classes of schedulers will thus arguably be (more) useful. However, as long as continuous information is involved (such as the values v), some form of discretisation of the state space

Table 3. Results ($\hat{p}_{\max}$) for SA

class	q	M_1	M_2	M_3	M_4	M_5	M_6
$\mathfrak{S}^{hist}_{\ell,v,e}$	1	0.50					
	2	**0.75**					
	4	**0.75**					
$\mathfrak{S}^{hist}_{\ell,v,o}$	1	0.50					
	2	0.50					
	4	0.50					
$\mathfrak{S}^{hist}_{\ell,t,e}$	1			**1.00**			
	2			0.86			
	4			0.62			
$\mathfrak{S}^{hist}_{\ell,\text{-},o}$	1		**0.90**				
	2		0.83				
	4		0.75				
$\mathfrak{S}^{hist}_{\ell,v,\text{-}}$	1						1.00
	2						1.00
	4						1.00
$\mathfrak{S}^{hist}_{\ell,t,\text{-}}$	1			**1.00**			
	2			0.87			
	4			0.78			
$\mathfrak{S}^{ml}_{\ell,v,e}$	1	0.50		0.50			
	2	**0.75**		0.81			
	4	**0.75**		**0.81**			
$\mathfrak{S}^{ml}_{\ell,v,o}$	1	0.50	**0.82**				
	2	0.50	0.79				
	4	0.50	0.71				
$\mathfrak{S}^{ml}_{\ell,t,e}$	1	0.50		0.50	**0.87**		
	2	**0.75**		**0.64**	0.64		
	4	**0.75**		0.60	0.54		
$\mathfrak{S}^{ml}_{\ell,t,o}$	1	0.50		**1.00**		0.83	
	2	0.50		**1.00**		0.84	
	4	0.50		0.97		**0.85**	
$\mathfrak{S}^{ml}_{\ell,\text{-},e}$	1	0.50		0.50	**0.71**		
	2	**0.75**		0.62	0.66		
	4	**0.75**		**0.66**	0.56		
$\mathfrak{S}^{ml}_{\ell,\text{-},o}$	1	0.50		1.00		0.83	
	2	0.50		1.00		0.83	
	4	0.51		1.00		0.83	
$\mathfrak{S}^{ml}_{\ell,v,\text{-}}$	1			0.51			0.50
	2			0.63			0.50
	4			**0.78**			0.50
$\mathfrak{S}^{ml}_{\ell,t,\text{-}}$	1			0.50	0.71		
	2			0.50	**0.77**		
	4			0.50	**0.77**		
$\mathfrak{S}^{ml}_{\ell,\text{-},\text{-}}$	1				0.71		
	2				0.71		
	4				0.71		

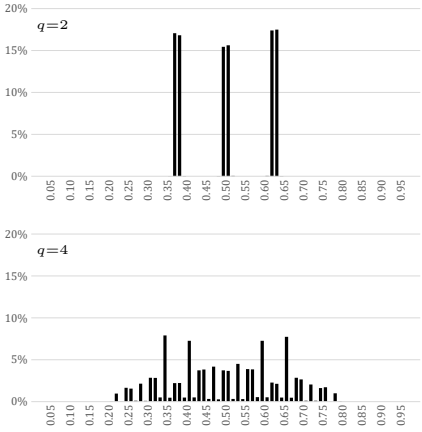

Fig. 13. Histograms for M_3 and $\mathfrak{S}^{ml}_{\ell,v,\text{-}}$

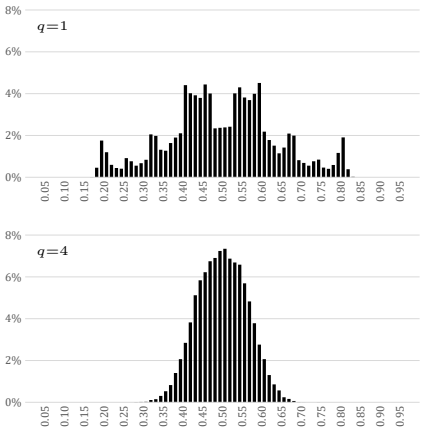

Fig. 14. Histograms for M_2 and $\mathfrak{S}^{ml}_{\ell,v,o}$

is needed. As we show below, there is ample room for the development of good discretisations and exploitation of the tradeoffs between classes in LSS for SA.

5.3 Experiments

We have implemented a prototype of LSS for SA in modes. It performs simulation on the exact concrete state space, but provides to schedulers a discretised view: for each real-valued quantity, we identify all values in the same interval $[\frac{i}{q}, \frac{i+1}{q})$, for integers i, q. We report experimental results on M_1 (Fig. 11) and M_2 through M_6 (see [8]) using LSS with $m = 10000$ and $n = 14889$ (so that $\epsilon = 0.005$, cf. Sect. 3.2) for each set of scheduler classes distinguished by the respective SA and

discretisation factors $q \in \{1, 2, 4\}$. All models have a structure similar to M_1, and in Table 3 we show the estimated lower bounds on the max. probabilities $\hat{p}_{\max}$ of reaching ✓. We highlight the best result among the discretisation factors.

Discussion. Increasing the discretisation factor or increasing the scheduler power generally also increases the number of decisions the schedulers *can* make. This may also increase the number of *critical* decisions a scheduler *must* make to achieve the max. probability. We clearly see this in the results. Some schedulers achieve the best probability only with the finest discretisation, indicating cases where fine discretisation is important for optimality and optimal schedulers inside the class are not rare. We show the histograms for M_3, $\mathfrak{S}^{ml}_{\ell,v,\text{-}}$, and $q \in \{2, 4\}$ in Fig. 13. Indeed many extremal schedulers are found, and the variance appears to depend only on the discretisation. On the other hand, some classes perform worse on some models as the discretisation gets finer, usually indicating that optimal schedulers are rare. Several other patterns exist, including a case where $q = 2$ yields the best results, clearly exhibiting the tradeoff between fine discretisation (i.e. a good scheduler is in the class) and rarity of near-optimal schedulers (i.e. the good schedulers will very rarely be sampled). However, these intuitions do not always match; one interesting case is $\mathfrak{S}^{ml}_{\ell,v,\text{-}}$ for M_2. Its expiration times are drawn from a wide range, up to $\textsc{Uni}(0, 8)$, compared to the other SA that use at most $\textsc{Uni}(0, 2)$. Thus $q = 1$ is already a relatively finer discretisation. Looking at the histograms in Fig. 14, we see that the spread of schedulers is good for $q = 1$, with schedulers on both ends of the spectrum being rather likely, and results being near the actual maximum probability of approx. 0.82. However, as the discretisation gets finer, the increase in the number of decisions dominates the potential to make better decisions, resulting in schedulers almost normally distributed around the "random guess" behaviour that leads to probability 0.5.

The experiments demonstrate that LSS can produce useful and informative results with SA, but that there is a lot of potential for better discretisations.

6 Conclusion

We have taken a tour through the opportunities and challenges in LSS on three continuous-time models. Thanks to discrete abstractions that fully preserve optimal probabilities developed for exact model checking, efficient techniques for PTA now exist. However, tackling time-bounded properties for MA, and any kind of efficient LSS at all for SA, remain open challenges. Our preliminary results for these two continuous-time and continuously stochastic models indicate that LSS shows potential for MA (as evidenced by our case studies), and offers a versatile tool to experiment with different restricted classes of schedulers on SA. We plan to develop better discretisations for SA, and apply LSS on larger case studies for both MA and SA. The ability to visualise the distribution of schedulers provides valuable insights into the character of a model's nondeterminism.

Acknowledgments. The authors thank Yuliya Butkova (Saarland University) for clarifying discussions on uniformisation and the time-bounded analysis of MA.

Experiment replication. We provide an artifact package [19] for independent replication of our experiments. It contains modes, all model files, the raw results, tabular views of those results (from which we derived tables 1 to 3 and the histograms), and the Linux shell scripts that we used to perform the experiments.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
2. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets, Advances in Petri Nets. Lecture Notes in Computer Science, vol. 3098, pp. 87–124. Springer (2003)
3. Bohlender, D., Bruintjes, H., Junges, S., Katelaan, J., Nguyen, V.Y., Noll, T.: A review of statistical model checking pitfalls on real-time stochastic models. In: ISoLA. Lecture Notes in Computer Science, vol. 8803, pp. 177–192. Springer (2014)
4. Brázdil, T., Chatterjee, K., Chmelik, M., Forejt, V., Kretínský, J., Kwiatkowska, M.Z., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: ATVA. Lecture Notes in Computer Science, vol. 8837, pp. 98–114. Springer (2014)
5. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Better automated importance splitting for transient rare events. In: SETTA. Lecture Notes in Computer Science, vol. 10606, pp. 42–58. Springer (2017)
6. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: A statistical model checker for nondeterminism and rare events. In: TACAS. Lecture Notes in Computer Science, vol. 10806. Springer (2018)
7. Butkova, Y., Hatefi, H., Hermanns, H., Krcál, J.: Optimal continuous time Markov decisions. In: ATVA. Lecture Notes in Computer Science, vol. 9364, pp. 166–182. Springer (2015)
8. D'Argenio, P.R., Gerhold, M., Hartmanns, A., Sedwards, S.: A hierarchy of scheduler classes for stochastic automata. In: FoSSaCS. Lecture Notes in Computer Science, vol. 10803. Springer (2018)
9. D'Argenio, P.R., Hartmanns, A., Legay, A., Sedwards, S.: Statistical approximation of optimal schedulers for probabilistic timed automata. In: iFM. Lecture Notes in Computer Science, vol. 9681, pp. 99–114. Springer (2016)
10. D'Argenio, P.R., Katoen, J.P.: A theory of stochastic systems part I: stochastic automata. Inf. Comput. 203(1), 1–38 (2005)
11. D'Argenio, P.R., Legay, A., Sedwards, S., Traonouez, L.M.: Smart sampling for lightweight verification of Markov decision processes. Software Tools for Technology Transfer 17(4), 469–484 (2015)
12. David, A., Jensen, P.G., Larsen, K.G., Mikucionis, M., Taankvist, J.H.: Uppaal Stratego. In: TACAS. Lecture Notes in Computer Science, vol. 9035, pp. 206–211. Springer (2015)
13. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 349–355. Springer (2011)
14. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS. pp. 342–351. IEEE Computer Society (2010)
15. Fehnker, A., Chaudhary, K.: Twenty percent and a few days – optimising a Bitcoin majority attack. In: NASA Formal Methods. Lecture Notes in Computer Science, vol. 10811, pp. 157–163. Springer (2018)

16. Forejt, V., Kwiatkowska, M.Z., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: SFM. Lecture Notes in Computer Science, vol. 6659, pp. 53–113. Springer (2011)
17. Guck, D., Hatefi, H., Hermanns, H., Katoen, J.P., Timmer, M.: Modelling, reduction and analysis of Markov automata. In: QEST. Lecture Notes in Computer Science, vol. 8054, pp. 55–71. Springer (2013)
18. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods in System Design 43(2), 191–232 (2013)
19. Hartmanns, A.: Lightweight statistical model checking in nondeterministic continuous time (artifact). 4TU.Centre for Research Data (2018), http://doi.org/10.4121/uuid:1453a13b-10ae-418f-a1ae-4acf96028118
20. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: TACAS. Lecture Notes in Computer Science, vol. 8413, pp. 593–598. Springer (2014)
21. Hartmanns, A., Hermanns, H., Krcál, J.: Schedulers are no prophets. In: Semantics, Logics, and Calculi. Lecture Notes in Computer Science, vol. 9560, pp. 214–235. Springer (2016)
22. Hartmanns, A., Sedwards, S., D'Argenio, P.R.: Efficient simulation-based verification of probabilistic timed automata. In: Winter Simulation Conference. pp. 1419–1430. IEEE (2017)
23. Hatefi, H., Hermanns, H.: Model checking algorithms for Markov automata. Electronic Communications of the EASST 53 (2012)
24. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI. Lecture Notes in Computer Science, vol. 2937, pp. 73–84. Springer (2004)
25. Kearns, M.J., Mansour, Y., Ng, A.Y.: A sparse sampling algorithm for near-optimal planning in large Markov decision processes. Machine Learning 49(2-3), 193–208 (2002)
26. Kroese, D.P., Nicola, V.F.: Efficient estimation of overflow probabilities in queues with breakdowns. Performance Evaluation 36, 471–484 (1999)
27. Kurkowski, S., Camp, T., Colagrosso, M.: MANET simulation studies: the incredibles. Mobile Computing and Communications Review 9(4), 50–61 (2005)
28. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer (2011)
29. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theor. Comput. Sci. 282(1), 101–150 (2002)
30. Legay, A., Sedwards, S., Traonouez, L.M.: Scalable verification of Markov decision processes. In: WS-FMDS at SEFM. Lecture Notes in Computer Science, vol. 8938, pp. 350–362. Springer (2014)
31. Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. Annals of the Institute of Statistical Mathematics 10(1), 29–35 (1959)
32. Reijsbergen, D., de Boer, P., Scheinhardt, W.R.W., Haverkort, B.R.: On hypothesis testing for statistical model checking. Software Tools for Technology Transfer 17(4), 377–395 (2015)
33. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: CAV. Lecture Notes in Computer Science, vol. 2404, pp. 223–235. Springer (2002)