International Journal on Software Tools for Technology Transfer
https://doi.org/10.1007/s10009-020-00563-2

STTT

Special Issue TACAS 2018

An efficient statistical model checker for nondeterminism and rare events

Carlos E. Budde[1] · Pedro R. D'Argenio[2,3,4] · Arnd Hartmanns[1] · Sean Sedwards[5]

Abstract

Statistical model checking avoids the state space explosion problem in verification and naturally supports complex non-Markovian formalisms. Yet as a simulation-based approach, its runtime becomes excessive in the presence of rare events, and it cannot soundly analyse nondeterministic models. In this article, we present modes: a statistical model checker that combines fully automated importance splitting to estimate the probabilities of rare events with smart lightweight scheduler sampling to approximate optimal schedulers in nondeterministic models. As part of the MODEST TOOLSET, it supports a variety of input formalisms natively and via the JANI exchange format. A modular software architecture allows its various features to be flexibly combined. We highlight its capabilities using experiments across multi-core and distributed setups on three case studies and report on an extensive performance comparison with three current statistical model checkers.

1 Introduction

Statistical model checking (SMC [1,49,81]) is a formal verification technique for stochastic systems. Using a formal stochastic model, specified as e.g. a continuous-time Markov chain (CTMC) or a stochastic variant of Petri nets, SMC can answer questions such as "what is the probability of system failure between two inspections" or "what is the expected time to complete a given workload". SMC is gaining popularity for complex applications where traditional exhaustive probabilistic model checking is limited by the state space explosion problem and by the inability to efficiently handle non-Markovian formalisms or complex continuous dynamics. At its core, SMC is the integration of classical Monte Carlo simulation with formal models. By only sampling concrete traces of the model's behaviour, its memory usage is effectively constant in the size of the state space, and it is applicable to any behaviour that can effectively be simulated. However, its use in formal verification faces two key challenges: *rare events* and *nondeterminism*.

The result of an SMC analysis is an *estimate* $\hat{q}$ of some actual quantity of interest q together with a statement on the potential statistical error. A typical guarantee is that, with probability δ, any $\hat{q}$ will be within $\pm\epsilon$ of q. To strengthen such a guarantee, i.e. increase δ or decrease ϵ, more samples (that is, simulation runs) are needed. Compared to exhaustive model checking, SMC thus trades memory usage for accuracy or runtime. A particular challenge thus lies in rare events, i.e. behaviours of very low probability. Meaningful estimates need a small *relative* error: for a probability on the order of 10^{-19}, for example, ϵ should reasonably be on the order of 10^{-20}. In a standard Monte Carlo approach, this would require infeasibly many simulation runs.

The authors are listed in alphabetical order. This work was supported by ANPCyT Project PICT-2017-3894 (RAFTSys), by ERC Grant 695614 (POWVER), by the JST ERATO HASUO Metamathematics for Systems Design Project (JPMJER1603), by the NWO SEQUOIA Project, by NWO VENI Grant 639.021.754, and by SeCyT-UNC Project 33620180100354CB (ARES).

✉ Arnd Hartmanns
a.hartmanns@utwente.nl

Carlos E. Budde
c.e.budde@utwente.nl

Pedro R. D'Argenio
dargenio@famaf.unc.edu.ar

Sean Sedwards
sean.sedwards@uwaterloo.ca

1 University of Twente, Enschede, The Netherlands

2 Universidad Nacional de Córdoba, Córdoba, Argentina

3 CONICET, Córdoba, Argentina

4 Saarland University, Saarbrücken, Germany

5 University of Waterloo, Waterloo, Canada

Published online: 28 May 2020

SMC naturally works for formalisms with non-Markovian behaviour and complex continuous dynamics for which the exact model checking problem is intractable or undecidable, such as generalised semi-Markov processes (GSMP) and stochastic hybrid Petri nets with many generally distributed transitions [67]. As a simulation-based approach, however, SMC is incompatible with nondeterminism. Yet (continuous and discrete) nondeterministic choices are desirable in formal modelling for concurrency, abstraction, and to represent either controllable inputs or an absence of knowledge. They occur in many formalisms such as Markov decision processes (MDP [68]) or probabilistic timed automata (PTA [58]). In the presence of nondeterminism, quantities of interest are defined with respect to optimal *schedulers* (also called policies, adversaries, or strategies) that resolve all nondeterministic choices: the verification result is the *maximum* or *minimum* probability or expected value ranging over *all* schedulers. Many SMC tools that appear to support nondeterministic models as input, e.g. PRISM [57] and UPPAAL SMC [26], use a single implicit scheduler by resolving all choices randomly. Results are thus only guaranteed to lie *somewhere* between minimum and maximum. Such implicit resolutions are a known problem affecting the trustworthiness of simulation studies [56].

In this article, we present modes, a statistical model checker that addresses both of the above challenges: It implements *importance splitting* [59] to efficiently estimate the probabilities of rare events, and *lightweight scheduler sampling* [60] to statistically approximate optimal schedulers. Both methods can be combined to perform rare event simulation for nondeterministic models.

Rare event simulation A key challenge in rare event simulation (RES [72]) is to achieve a high degree of automation for a general class of models [8,51,74,82]. For this purpose, following the reasoning in [10], we focus on importance splitting RES algorithms. Current approaches to automatically derive the importance function for importance splitting, which is critical for the method's performance, are mostly limited to restricted classes of models and properties, e.g. [36,62,75]. modes combines several importance splitting techniques with the compositional importance function construction of Budde et al. [12] and two different methods to derive levels and splitting factors [9]. These method combinations apply to arbitrary stochastic models with a partly discrete state space. We have shown them to work well across different Markovian and non-Markovian automata- and dataflow-based formalisms [9]. We present details on modes' support for RES in Sect. 3. Alongside PLASMA LAB [61], which implements automatic *importance sampling* [53] and semi-automatic importance splitting [52,54] for Markov chains (with APIs allowing for extensions to other models), modes is one of the most automated tools for RES on formal models today. In particular, we are not aware of any other tool that provides fully automated RES on general stochastic models.

Nondeterminism Sound SMC for nondeterministic models is a difficult problem. For MDP, Brázdil et al. [7] proposed a sound machine learning technique to incrementally improve a partial scheduler. UPPAAL STRATEGO [25] 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 as all scheduler decisions must be stored explicitly. Classic memory-efficient sampling approaches like the one of Kearns et al. [55] address discounted models only. modes implements the lightweight scheduler sampling (LSS) approach introduced by Legay et al. [60]. It is currently the only technique that applies to reachability probabilities and undiscounted expected rewards—as typically considered in formal verification—that also keeps memory usage effectively constant in the number of states. Its efficiency depends only on the likelihood of sampling near-optimal schedulers. modes implements the existing LSS approaches for MDP [60] and PTA [21,46], for unbounded properties on Markov automata (MA [30]) and provides prototypical support [22] for LSS with different scheduler classes [20] on stochastic automata (SA [23]). We describe modes' LSS implementation in Sect. 4.

The modes tool modes is part of the MODEST TOOLSET [43], which also includes the explicit-state model checker mcsta and the model-based tester motest [38]. It inherits the toolset's support for a variety of input formalisms, including the high-level process algebra-based MODEST language [39] and xSADF [44], an extension of scenario-aware dataflow. Many other formalisms are supported via the JANI interchange format [13]. As simulation is easily and efficiently parallelisable, modes fully exploits multi-core systems, but can also be run in a distributed fashion across homogeneous or heterogeneous clusters of networked systems. We describe the various methods implemented to make modes a correct and scalable statistical model checker that supports classes of models ranging from discrete-time Markov chains (DTMC [4]) to stochastic hybrid automata (SHA [32]) in Sect. 2. We focus on its software architecture in Sect. 5, explaining how its flexibility and modularity make it easy to combine the various individual techniques to obtain, for example, distributed rare event simulation with scheduler sampling for expected rewards, and how new techniques, types of models, or measures of interest can be added. Finally, we provide an evaluation of its features, flexibility, and performance in Sects. 6, and 7: we first use three very different case studies to highlight the varied kinds of models and analyses that modes can handle and how it enables entirely new types of analyses; we then compare the performance of its Monte Carlo and rare-event simulation engines to PLASMA LAB, PRISM, and FIG [8].

Previous publications. modes was first described in a tool demonstration paper in 2012 [5]. At that time, its focus was on the use of partial order and confluence reduction-based techniques [47] to decide on-the-fly if the nondeterminism in a model is spurious, i.e. whether maximum and minimum values are the same and an implicit randomised scheduler can safely be used. modes was again mentioned as a part of the MODEST TOOLSET in 2014 [43]. Since then, modes has been completely redesigned. The partial order and confluence-based methods have been replaced by LSS, enabling the simulation of non-spurious nondeterminism; automated importance splitting has been implemented for rare event simulation; support for MA and SHA has been added; the statistical evaluation methods have been extended and improved. Concurrently, advances in the shared infrastructure of the MODEST TOOLSET, now at version 3, provide access to new modelling features and formalisms as well as support for the JANI specification.

This article is an extended version of a conference tool paper on modes [11]. We have expanded Sect. 2.1 with examples and give a more detailed description of the differences between the simulation algorithms implemented in modes. We implemented a new statistical evaluation method and significantly extended the corresponding Sect. 2.3. Section 3 includes additional explanations and figures. In Sect. 4, we have added a description and illustrations of the new scheduler histograms feature first introduced in [22]. Finally, as suggested by the conference paper's reviewers, we have performed a systematic performance comparison with other statistical model checkers. We report on the results in Sect. 7, which now complements the modes-only experiments of Sect. 6 (the purpose of which is to highlight the features and versatility of the tool by itself).

2 Ingredients of a statistical model checker

A statistical model checker performs a number of tasks to analyse a given formal model with respect to to a property of interest. First, it needs to simulate a model, i.e. generate random samples of its behaviour and determine the value of the property for each of the samples. This value will typically be 1 or 0 when estimating a probability, but can be an arbitrary (real or, in practice, floating-point) number for expected rewards. It must then perform a statistical evaluation of the sampled values, either on-the-fly or after a certain number of samples have been generated, to determine when or if there is enough evidence to return a result for the property with the desired statistical error and confidence. Sample generation is trivially parallelisable, but in doing so, the statistical model checker must take care to avoid introducing a bias into the evaluation. In this section, we describe how modes implements these tasks and addresses their inherent challenges. All random selections in an SMC tool are typically resolved by a *pseudo*-random number generator (PRNG). modes implements several different PRNGs; it uses the well-established "Mersenne Twister MT19937" PRNG [64] by default. For brevity, we write "random" to mean "pseudo-random" in this section.

2.1 Simulating different model types

The most basic task of a statistical model checker is *simulation*: the generation of random samples—*simulation runs*—from the probability distribution over behaviours defined by the model. The complexity of this task inherently depends on the model type: Simulating a DTMC is conceptually simple, but accurately simulating a stochastic hybrid system with complex nonlinear dynamics requires advanced techniques to e.g. make sure that no discrete events whose timing depends on the evolution of the continuous quantities are skipped. modes uses the infrastructure of the MODEST TOOLSET to transform various input languages into an internal representation corresponding to a network of stochastic hybrid automata (SHA [39]) with discrete variables. The representation directly corresponds to a JANI model, compactly representing a large or infinite state space. It is then compiled into bytecode implementing a low-level interface to explore the concrete state space, which modes shares with the mcsta model checker. Based on this interface to the compiled model, modes contains simulation algorithms specifically optimised for the different types of models. As the model types get more complex, so do the algorithms. modes' simulation runtime in practice is thus higher for the more complex model types, especially for SHA. We thus need to use the most specialised simulation algorithm for each model. We graphically contrast the model types with dedicated support in modes in Figs. 1 and 2.

2.1.1 DTMC and MDP

DTMC and MDP are discrete-time models, i.e. there is no notion of continuous time; a simulation run moves from state to state in discrete steps. The successor state of each step is chosen according to a discrete probability distribution. In MDP, but not in DTMC, a state may provide multiple outgoing transitions (usually distinguished by different action labels such as $\mathtt{a}$, $\mathtt{b}$, and τ in Fig. 1), representing nondeterministic choices. Simulation for this type of model is simple and efficient, and modes implements a single simulation algorithm to cover both DTMC and MDP[1]:

[1] Here and in the following subsections, we assume that models are free of deadlocks and timelocks for clarity of presentation; modes does correctly simulate models with deadlocks or timelocks.

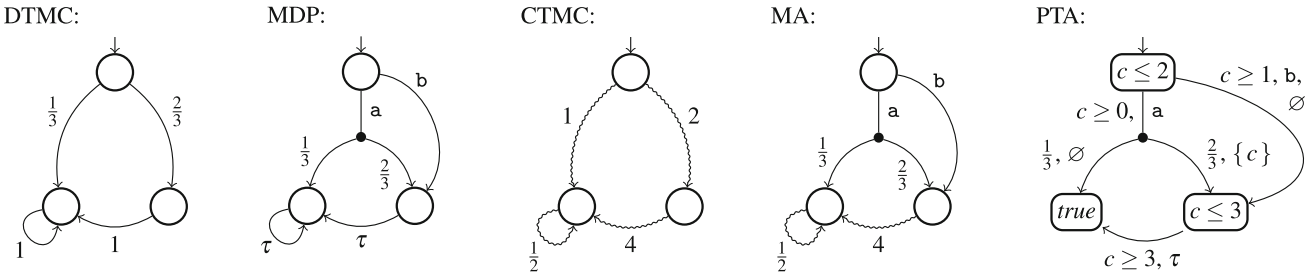

Fig. 1 Examples of different model types supported by modes

1. If the value of the property can be decided in the current state at the current step count: return that value.
2. Obtain the current state's transitions. DTMC will only have one transition in this step.
3. Use LSS (Sect. 4) to select one of the transitions.
4. Use the transition's probability distribution to randomly select a successor state.
5. Increment the step count and continue from the successor state.

2.1.2 MA and CTMC

CTMC and MA are stochastic continuous-time models. In CTMC, every transition is annotated with a rate. Let λ be the sum of the rates of all outgoing transitions of a state: the state's *exit rate*. The time spent in that state then follows a (negative) exponential distribution with rate λ, i.e. the probability to spend at most t time units is $1 - e^{-\lambda t}$. After that time, one transition is chosen randomly—the probability of a transition with rate λ_t being $\frac{\lambda_t}{\lambda}$—and the CTMC moves to the transition's target state. In the models of Fig. 1, we highlight transitions with rates by squiggly lines. MA add a second type of transitions that behave like those in MDP. The time spent in a state that has at least one such transition is always zero; thus, the probability of choosing a transition with a rate out of such a state is also zero. Transitions with rates are called *Markovian*, while those with actions as in MDP are called *immediate* transitions. CTMC and MA are covered by one simulation algorithm in modes, which is slightly more involved than the one for MDP due to the need to manage two types of transition and keep track of continuous time:

1. If the value of the property can be decided in the current state at the current step count and total elapsed time t, return that value.
2. Obtain the current state's transitions and separate them into Markovian and immediate transitions. CTMC will only have Markovian transitions in this step.
3. If there is at least one immediate transition:
   1. Use LSS (Sect. 4) to select one of them.
   2. Use the transition's probability distribution to randomly select a successor state.

   Otherwise, if there are only Markovian transitions:
   1. Sample a value t' from the exponential distribution parameterised by the current state's exit rate.
   2. If the value of the property can be decided in the current state at the current step count and at any point between total elapsed times t and $t + t'$, return that value.
   3. Increase t by t'.
   4. Pick one of the transitions randomly, using the rates as probability weights. Its target is the successor state.
4. Increment the step count and continue from the successor state.

The algorithm relies on the memoryless property of the exponential distribution: there is no need to keep a memory of time (e.g. via clocks as in PTA and SHA) beyond the total elapsed time to support time-bounded properties.

2.1.3 Probabilistic timed automata

Probabilistic timed automata (PTA [58]) extend MDP with clock variables, edge guards, and location invariants as in timed automata. Like MA, they are a continuous-time model, but explicitly keep a memory of elapsed times in the clocks. Due to the presence of variables and expressions, we say that PTA are a symbolic model; a PTA thus consist of *locations* and *edges*. Its semantics is an MDP-like object with uncountably many *states* and *transitions*; each state consists of the current location and a valuation that assigns concrete values to all clock variables. PTA admit finite-state abstractions that preserve reachability probabilities and allow them to essentially be simulated as MDP. modes implements two dedicated simulation algorithms for PTA based on the region [21] and zone graph [46] abstractions. These abstractions do not preserve rewards, and the algorithms are computationally much more involved than the ones for DTMC/MDP and CTMC/MA. However, by performing simulation for the continuous-time model of PTA on an entirely finite abstrac-

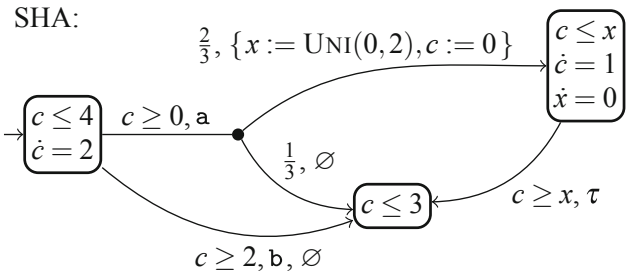

Fig. 2 An example of the SHA model type

tion, they enable effective LSS for PTA (Sect. 4). With fewer restrictions, PTA can also be treated as SHA whenever LSS is not needed:

2.1.4 Stochastic timed and hybrid automata

SHA extend PTA with general continuous probability distributions and continuous variables with dynamics governed by differential equations and inclusions. We show a simple example SHA in Fig. 2. If the differential equation for variable c is $\dot{c} = 1$ in all locations, i.e. if c is a clock, this SHA would be a stochastic timed automaton (STA [6]).

modes implements a simulation algorithm for *deterministic* SHA where all differential equations are of the form $\dot{v} = e$ for a continuous variable v and an expression e over *discrete* variables. This subset can be simulated without the need for approximations; it corresponds to deterministic rectangular hybrid automata [48]. For each transition, the SHA simulator needs to compute the set of time points at which it is enabled. These sets can be unions of several disjoint intervals, which results in relatively higher computational effort for SHA simulation. Furthermore, since SHA may use general probability distributions to control the passage of time (the SHA of Fig. 2, for example, uses a continuous uniform distribution to determine the amount of time spent in the rightmost location), all sampled values plus the values of all clocks need to be stored and updated individually.

The SHA simulation algorithm operates on the semantics where states are pairs of a location and a valuation for all variables like in PTA. It proceeds as follows:

1. If the value of the property can be decided in the current state at the current step count and total elapsed time t, return that value.
2. Compute the set of delays after which the current location's invariant is still satisfied, and for each available edge the set of delays after which the edge's guard is enabled.
3. Pick a delay t such that the invariant is continuously satisfied and at least one edge is enabled. (If more than one such delay exists, the SHA is nondeterministic, and we abort simulation with an error message.)
4. If the value of the property can be decided in the current state at the current step count and at any point between total elapsed times t and $t + t'$, return that value.
5. Increase t by t' and update the values of all continuous variables in the state's valuation according to their differential equations for the passage of t' time units.
6. If more than one edge is now enabled, either abort due to nondeterminism, or use the LSS prototype implementation for SA (Sect. 4) to select one of them.
7. Use the edge's symbolic probability distribution evaluated in the current state to randomly select a successor.
8. Increment the step count and continue from the successor state.

2.2 Properties and termination

SMC computes a value for the property on every simulation run. A run is a finite trace; consequently, standard SMC only works for linear-time properties that can be decided on finite traces. modes supports two classes of properties: transient properties and expected rewards. They can come as *queries* for the concrete value (i.e. for the probability or the expected reward) or as *requirements* that compare the value to a bound. Every query q can be turned into a requirement $q \sim c$ by adding a comparison $\sim \in \{\leq, \geq\}$ to a constant value $c \in \mathbb{R}$.

2.2.1 Transient properties

Transient (reachability) queries are of the form $\mathbb{P}(\neg \mathit{avoid}\ \mathtt{U}\ \mathit{goal})$ for the probability of reaching a set of states characterised by the state formula *goal* before entering the set of states characterised by state formula *avoid*. A state formula is an expression over the (discrete and continuous) variables of the model without any temporal operators. Transient queries may be step-, time-, and reward-bounded. An example transient query is "what is the probability to reach a destination (*goal*) within an energy budget (a reward bound) while avoiding collisions (*avoid*)".

A simulation run ends when the value of a property is decided. For transient properties, this is the case when reaching a *goal* state (value 1), and when entering an *avoid* state, encountering a deadlock, or violating a step, time, or reward bound (value 0). To ensure termination, the probability of eventually encountering one of these events must be 1. modes additionally implements cycle detection: it keeps track of a configurable number n of previous visited states. When a run returns to a previous state without intermediate steps of probability < 1, it will loop forever on this cycle and the run has value 0. modes uses $n = 1$ by default for good performance while still allowing models built for model checking, which avoid deadlocks but often contain

terminal states with self-loops, to be simulated. Techniques to detect bottom strongly connected components on-the-fly [19] would enable SMC for unbounded linear-time properties on Markov chains that do not conform to this requirement, but they require additional information about the state space and are not yet implemented in modes.

2.2.2 Expected rewards

Expected reward queries are of the form $\mathbb{E}(\mathit{reward} \mid \mathit{goal})$ for the expected accumulated reward (or cost) over the reward structure *reward* when reaching a location in the set of states characterised by *goal* for the first time. A reward structure assigns a rate reward $r(s) \in \mathbb{R}$ to every state s and a branch reward $r(b) \in \mathbb{R}$ to every probabilistic branch b of every transition. Expected reward queries allow asking for e.g. the expected number of retransmissions (the reward) until a message is successfully transmitted (*goal*) in a wireless network protocol.

For expected rewards, when entering a *goal* state, the property is decided with the value being the sum of the rewards along the run. By definition [31, Section 5.3], when a run enters a deterministic cycle, an expected-reward property is decided with value ∞. One of these situations—reaching a *goal* state or entering a deterministic cycle—must occur with probability 1 to ensure termination for expected rewards. Models built for model checking almost always have this property, since otherwise the expected reward would be ∞ by definition and thus not of any particular interest.

2.3 Statistical evaluation of samples

Simulating n runs provides a sequence of independent values $v_1, \ldots, v_n$ for the property. $\hat{v}_n = \frac{1}{n}\sum_{i=1}^{n} v_i$ is an unbiased estimator of the actual probability or expected reward v. An SMC tool must stop generating runs at some point, and quantify the statistical properties of the estimate $\hat{v} = \hat{v}_n$ returned to the user. modes implements four different methods for this purpose. All methods can be configured with three common parameters:

- n: the number of simulation runs (unspecified by default),
- δ: the level of "confidence" (0.95 by default), and
- ϵ: the precision or half-width parameter, which can be requested as absolute or relative precision (the latter denoted "$\times\epsilon$", with defaults of 0.01 and 10 %, respectively).

All methods require exactly one of the parameters to be unspecified; the admissible combinations of parameters depend on the method.

A priori, the outcome of the i-th simulation run is a random variable X_i. For transient properties using standard Monte Carlo simulation, it is Bernoulli-distributed; for transient properties using rare event simulation (Sect. 3) and for expected-reward properties, it follows an unknown distribution over $[0, \infty)$. Whether a statistical method as implemented in modes can be applied to a model and property depends on the distribution of the X_i, on whether the property is a query or a requirement, and on which parameter is left unspecified. We summarise these dependencies in Table 1; bold entries mark the default method chosen by modes in the specific situation unless another method is explicitly requested by the user. We now provide a brief description of each method; for a broader overview of statistical methods and especially hypothesis tests for SMC, we refer the interested reader to [70].

Table 1 The statistical evaluation methods implemented in modes

Parameter values given	$\{0, 1\}$ transient (MC)		$[0, \infty)$ transient (RES), rewards	
	Query	Requirement	Query	Requirement
n, δ	**Okamoto** CI (binom.)	**Okamoto** CI (binom.)		
n, ϵ	**Okamoto**	**Okamoto**		
δ, ϵ	**Adaptive** Okamoto CI (binom.)	**SPRT** Adaptive Okamoto CI (binom.)		**CI (CLT)**
$\delta, \times\epsilon$	**Adaptive** CI (binom.)	**Adaptive** CI (binom.)		

2.3.1 Confidence intervals

Confidence intervals are likely the most widely used method to quantify the statistical properties of a probabilistic experiment. modes' **CI** method returns a confidence interval $[x, y]$ that contains $\hat{v}$, with $y - x = 2 \cdot \epsilon$. Its guarantee is that, if the SMC analysis is repeated many times, $100 \cdot \delta$ % of the confidence intervals will contain v. For Bernoulli-distributed X_i, modes constructs a binomial proportion confidence interval. It uses the "exact" Clopper–Pearson interval [18,73] for $\hat{v} \in \{0, 1\}$ and the Agresti–Coull approximation [2] otherwise. When the underlying distribution is unknown, modes uses the standard normal (or Gaussian) confidence interval. This relies on the central limit theorem for means, assuming a "large enough" n. modes requires $n \geq 50$ as a heuristic. Except for Clopper–Pearson, the computed interval is symmetric, i.e. $x = \hat{v} - w$. modes requires the user to specify δ plus either of ϵ, and n.

If n is not specified, the CI method becomes a sequential procedure: generate runs until the width of the interval for confidence δ is below $2 \cdot \epsilon$. This is the "Chow–Robbins" method [17], which, however, has only been proven to

guarantee confidence δ asymptotically as ϵ goes to 0. A corresponding warning message is generated whenever the Chow–Robbins method is used. When n is not specified, modes can also be instructed to interpret the value of ϵ as a relative half-width, i.e. the final interval will have width $\hat{v} \cdot 2 \cdot \epsilon$. While this is useful for rare events (and the only method for relative-width sequential estimation in the generally distributed case currently implemented), it is a "method of last resort" that is well-known not to guarantee the requested confidence [33, Section 3]. modes prints a more severe warning message than the one for the Chow–Robbins method in this case. The CI method can be turned into a hypothesis test for requirements $q \sim c$ by checking whether $\hat{v} \geq y$ or $\hat{v} \leq x$, and returning "undecided" if $\hat{v}$ is inside the interval.

Due to the various problems described above, confidence intervals are never chosen as a default by modes when another method can be used instead (cf. Table 1). In the generally distributed case, however, the CI method based on the central limit theorem assumption is the only method currently available.

2.3.2 The Okamoto bound

The **Okamoto** method, based on the Okamoto bound [66] (which is often referred to as the Chernoff-Hoeffding bound, and sometimes called the "APMC method" for the first SMC tool that implemented it [49]), guarantees for error ϵ and confidence δ that $\mathbb{P}(|\hat{v}-v| > \epsilon) < 1-\delta$. It only applies to the case of Bernoulli-distributed samples here. modes requires the user to specify any two of ϵ, δ and n, out of which the missing value is computed by solving the bound equation

$$n = \frac{\ln(\frac{2}{1-\delta})}{2 \cdot \epsilon^2}$$

accordingly (rounding up n to obtain an integer number of runs as necessary). Note that this means that the admissible parameter values are restricted such that $n \cdot (\epsilon^2) \geq \ln(2)/2$; modes checks that this is the case and otherwise auto-selects another method. The APMC method can be used as a hypothesis test for $\mathbb{P}(\cdot) \sim c$ by checking whether $\hat{v} \geq c + \epsilon$ or $\hat{v} \leq c - \epsilon$, and returning "undecided" if neither is true.

The main advantage of the Okamoto method—that any one missing parameter can be precomputed from the other two before simulation runs start—is also its main weakness: unless the true probability is close to 0.5, it requires far more runs than sequential methods that adapt n to the results of the runs as they come in. For this reason, the Okamoto method is used as the default only in those cases where the number of runs is explicitly specified by the user. In all other (Bernoulli) cases, it selects one of the two sound sequential methods presented below.

2.3.3 The new adaptive sampling method

The **Adaptive** method in modes implements the new adaptive sampling approach by Chen and Xu [16, Section III]. It is a sequential method, i.e. it requires n to be unspecified and performs simulation runs until a stopping criterion is met. The stopping criterion comes in two versions, one for absolute ϵ and one for relative error. The former provides the same guarantee as the Okamoto method, while the latter guarantees that $\mathbb{P}(|\hat{v}-v| > \epsilon \cdot v) < 1-\delta$. The key difference to the Okamoto method is that both stopping criteria take $\hat{v}_n$ into account. For example, the one for absolute ϵ is to keep generating runs as long as

$$n < \frac{2 \cdot \ln(\frac{2}{1-\delta})}{\epsilon^2} \cdot \left(\frac{1}{4} - \left(\left| \hat{v}_n - \frac{1}{2} \right| - \frac{2}{3} \cdot \epsilon \right)^2 \right).$$

In this way, the Adaptive method needs far fewer runs for the same ϵ and δ than the Okamoto method if v is far from 0.5. If v is close to 0.5, then it will require the same number of runs. The Adaptive method can be used for hypothesis testing to handle requirements in the same way as the Okamoto method.

The Adaptive method is the only one implemented in modes that provides guaranteed confidence for relative ϵ. Since it is also no worse than the Okamoto method in terms of the number of runs, modes chooses it as the default method for Bernoulli-distributed X_i when the number of runs is unspecified, except for the case of absolute ϵ for requirements, where the SPRT method is preferred (see below).

We have experimentally compared the number of runs required by the Okamoto and Adaptive methods on several of the DTMC models used in Sect. 7 with $\delta = 0.95$ and absolute $\epsilon = 0.001$. The results are shown in Table 2 (with "M" indicating millions of runs). For the Adaptive method, we report the averages over five independent invocations of modes. $\hat{v}$ is the average of the estimates reported for all six invocations. We used the same hardware as in Sect. 7, and multi-core simulation with 3 threads. We see that the Adaptive method indeed drastically reduces the number of runs needed, and consequently the simulation time, in those cases where the value is far from 0.5.

2.3.4 The sequential probability ratio test

modes also implements Wald's **SPRT**, the sequential probability ratio test [78]. As a sequential hypothesis test, it has no predetermined n, but decides on-the-fly whether more samples are needed as they come in, like the Adaptive method. It is a *test* for Bernoulli-distributed quantities, i.e. it only applies to transient requirements of the form $\mathbb{P}(\cdot) \sim c$ when analysed with standard Monte Carlo simulation. modes interprets ϵ as the indifference level parameter of the SPRT and sets its error

Table 2 The Adaptive and Okamoto methods compared

Model	Instance	$\hat{v}$	Okamoto		Adaptive	
			Runs	Time	Runs	Time
brp	16-2	0.00	1.84M	31 s	7221	1 s
	32-3	0.00		57 s	4980	1 s
crowds	3-5	0.05	1.84M	12 s	0.38M	4 s
	5-15	0.09		18 s	0.62M	7 s
	6-20	0.12		22 s	0.79M	10 s
egl	5-2	0.52	1.84M	31 s	1.84M	31 s
	10-6	0.50		137 s	1.84M	136 s
	20-8	0.50		347 s	1.84M	354 s
leader_sync	4-3	1.00	1.84M	5 s	4920	1 s
	5-4	1.00		5 s	5100	1 s
nand	20-2	0.42	1.84M	74 s	1.79M	66 s
	40-3	0.58		187 s	1.80M	176 s
	60-4	0.69		350 s	1.59M	293 s

parameter α to $1-\delta$. The SPRT method stops when the collected samples so far provide sufficient evidence to decide between $v \geq c+\epsilon$ or $v \leq c-\epsilon$ with probability $\leq \alpha$ of wrongly accepting either hypothesis. Note in particular that, in contrast to the hypothesis tests constructed from the previous methods, the SPRT has no "undecided" result; instead, if v is too close to c, it will randomly report the requirement as satisfied or unsatisfied.

The number of runs actually needed before the SPRT stops depends on the difference between the actual value v and the bound c; the larger it is, the sooner will the test conclude. The SPRT is optimal [79], i.e. there cannot be another sequential test that, for the same δ and ϵ, needs fewer runs on average. For this reason, modes uses the SPRT as the default for the one case where it is applicable (cf. Table 1).

2.4 Distributed sample generation

Simulation is easily and efficiently parallelisable. Yet a naïve implementation of the statistical evaluation—processing values from the runs in the order they flow in—risks introducing a bias in a parallel setting. Consider estimating the probability of system failure when simulation runs that encounter failure states are shorter than other runs, and thus quicker. In parallel simulation, failure runs will tend to arrive earlier and more frequently, thus overestimating the probability of failure. To avoid such bias, modes uses the adaptive schedule first implemented in YMER [80]. It works as follows, assuming simulation on n parallel nodes:

1. Initialise the schedule as queue $q = [1, \ldots, n]$. Create an empty queue of results q_i for each $i \in \{1, \ldots, n\}$.
2. Wait for the result r of a simulation run to arrive. Let i be the number of the node that generated r. Enqueue i in q and enqueue r in q_i.
3. Let $i = \text{front}(q)$. If q_i is empty, go back to step 2. Otherwise, dequeue i from q and let $r = \text{dequeue}(r)$. Process the result r and repeat step 3.

This method adapts to differences in the speed of nodes by scheduling to process more future results from fast nodes when current results come in quickly. It always commits to a schedule a priori before the actual results arrive, ensuring the absence of bias. In contrast to other methods such as the buffered fixed schedule of UPPAAL SMC [14], it is thus well-suited for heterogeneous clusters of machines with significant performance differences.

3 Automated rare event simulation

With the standard confidence of $\delta = 0.95$, we have $n \approx 1.84/\epsilon^2$ in the Okamoto method: for every decimal digit of precision, the number of runs increases by a factor of 100. If we attempt to estimate probabilities on the order of 10^{-4}, i.e. $\epsilon \approx 10^{-5}$, we need billions of runs and days or weeks of simulation time. This is the problem tackled by rare event simulation (RES) methods [72]. These increase the number of simulation runs that reach the rare event and adjust the statistical evaluation accordingly. The main RES methods are *importance sampling* and *importance splitting*. The former modifies the probability distributions that are part of the model, with the aim to make the event more likely to occur. The challenge lies in finding a "good" *change of measure* to modify probabilities in an effective way. Importance sam-

pling approaches are thus tailored to a specific type of model, and in particular mostly to different variants of and property types for Markov chains. Importance splitting instead does not modify the model, but rather changes the simulation algorithm to perform more (partial) simulation runs, which start from non-initial states and end early. Most importance splitting algorithms thus readily work for a wide range of different model types. Here, the challenge is to find an *importance function* $f_I : S \to \mathbb{N}$ that maps each state in S to its importance in $\{0, \ldots, \max f_I\}$: a value indicating how "close" it is to the rare event. More (partial) runs will be started from states with higher importance. The performance, but not the correctness, of all splitting methods hinges on the quality of the importance function.

modes implements RES for transient properties. Due to its focus on supporting different model types, including models with general probability distributions like SA and SHA, it uses importance splitting. modes implements recently developed methods to select all parameters of importance splitting, notably the importance function itself, in a fully automated way. We now give an overview of how these methods work, then present the three adjusted simulation algorithms that perform splitting in modes. For a more in-depth review of these techniques, we refer the interested reader to [10].

3.1 Deriving importance functions

Traditionally, the importance function is specified ad hoc by a RES expert [15,27,36,71,74,77]. Striving for usability by *domain* experts, modes implements the compositional importance function generation method of [12] that is applicable to any compositional stochastic model $\mathtt{M} = \mathtt{M}_1 \parallel \ldots \parallel \mathtt{M}_n$ with a partly discrete state space. We write $s|_i$ for the projection of state s of $\mathtt{M}$ to the discrete local variables of component $\mathtt{M}_i$. The method works as follows [9]:

1. Convert the goal set formula *goal* to negation normal form (NNF) and associate each literal $goal^j$ with the component $\mathtt{M}(goal^j)$ whose local state variables it refers to. Literals are required to not refer to multiple components.
2. Explore the *discrete part* of the state space of each component $\mathtt{M}_i$. For each $goal^j$ with $\mathtt{M}_i = \mathtt{M}(goal^j)$, use reverse breadth-first search to compute the local minimum distance $f_i^j(s|_i)$ of each state $s|_i$ to a state satisfying $goal^j$.
3. In the syntax of the NNF of *goal*, replace every occurrence of $goal^j$ by $f_i^j(s|_i)$ with i such that $\mathtt{M}_i = \mathtt{M}(goal^j)$, and every Boolean operator $\wedge$ or $\vee$ by $+$. Use the resulting formula as the importance function $f_I(s)$.

The method takes into account both the structure of the goal set formula and of the state space. This is in contrast to the approach of Jégourel et al. [52], implemented in a semi-automated fashion [54] in PLASMA LAB [61], that only considers the structure of the (more complex linear-time) property. The memory usage of the compositional method is determined by the number of discrete local states (required to be finite) over all components. Component state spaces are usually small even when the composed state space explodes combinatorially.

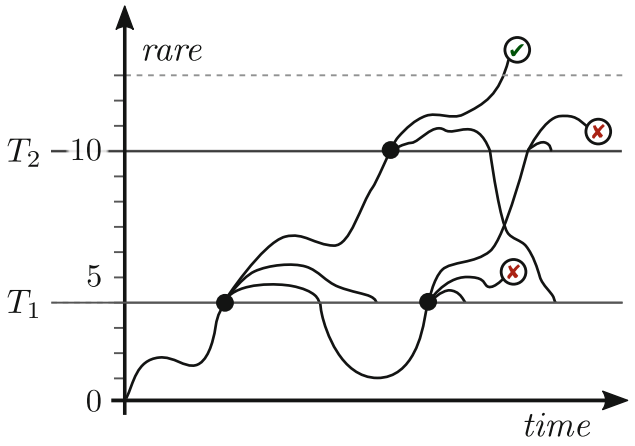

Fig. 3 Illustration of RESTART [9]

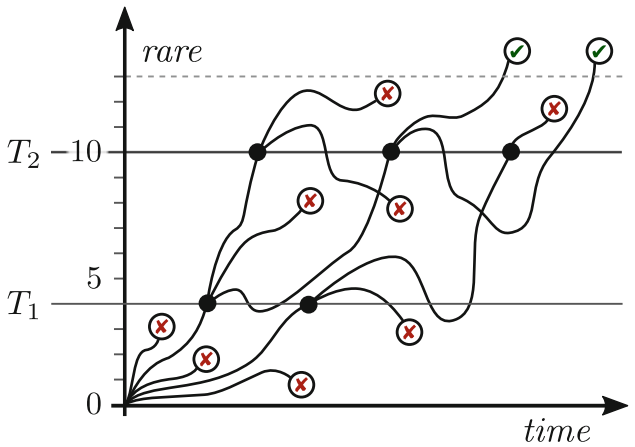

Fig. 4 Illustration of fixed effort [9]

3.2 Levels and splitting factors

We also need to specify *when* and *how much* to "split", i.e. increase the simulation effort. For this purpose, the values of the importance function are partitioned into *levels* and a *splitting factor* is chosen for each level [77]. Splitting too much too often will degrade performance (oversplitting), while splitting too little will cause starvation, i.e. few runs that reach the rare event. It is thus critical to choose good levels and splitting factors. Again, to avoid the user having to make these choices ad hoc, modes implements two methods to compute them automatically. One is based on the sequen-

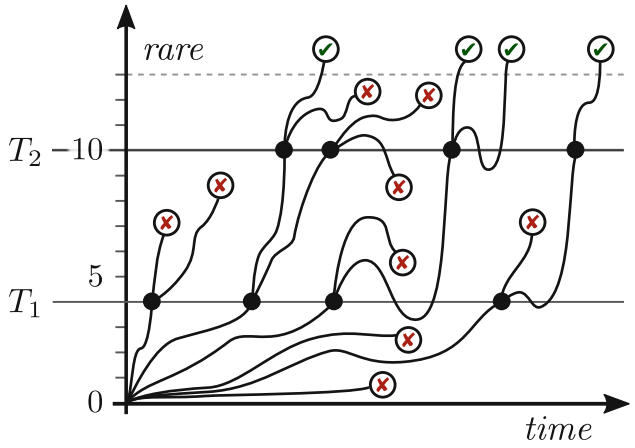

Fig. 5 Illustration of fixed success [9]

tial Monte Carlo splitting technique [15], while the other method, *expected success* [9], has been newly developed for modes. It strives to find levels and factors that lead to one run moving up from one level to the next in the expectation.

3.3 Importance splitting runs

The derivation of importance function, levels, and splitting factors is a preprocessing step. Importance splitting then replaces the simulation algorithm by a variant that takes this information into account to more often encounter the rare event. modes implements three importance splitting techniques: RESTART, fixed effort, and fixed success. They are implemented as wrappers around the simulation algorithms described in Sect. 2.1 and can be freely combined with any of them, i.e. with any model type supported by modes.

For all three methods, the average of the result of many runs is again an unbiased estimator for the probability of the transient property [34]. However, each run is no longer a Bernoulli trial. Of the statistical evaluation methods offered by modes, only CI with the central limit theorem assumption is thus applicable. For a deeper discussion of the challenges in the statistical evaluation of rare event simulation results, we refer the interested reader to [69]. To the best of our knowledge, modes is today the most automated rare event simulator for general stochastic models. In particular, it defaults to the combination of RESTART with the expected success method for level calculation, which has shown the most consistently good performance in [9].

3.3.1 Restart

RESTART [76] is illustrated in Fig. 3: As soon as a RESTART run crosses the threshold into a higher level, $n_\ell - 1$ new child runs are started from the first state in the new level, where n_ℓ is the splitting factor of level ℓ. When a run moves below its creation level, it ends. It also ends on reaching an *avoid* or *goal* state. The result of a RESTART run—consisting of a main and several child runs—is the number of runs that reach *goal* times $1/\prod_\ell n_\ell$, i.e. a rational number greater than or equal to zero.

3.3.2 Fixed effort

Runs of the *fixed effort* method [34,35], illustrated in Fig. 4, are rather different. They consist of a fixed number of partial runs on each level, each of which ends when it crosses into the next higher level or encounters a *goal* or *avoid* state. When all partial runs for a level have ended, the next round starts from the previously encountered initial states of the next higher level. When a fixed effort run ends, the fraction of partial runs started in a level that moved up approximates the conditional probability of reaching the next level given that the current level was reached. If *goal* states exist only on the highest level, the overall result is the product of all of these fractions, i.e. a rational number in the interval [0, 1].

3.3.3 Fixed success

Fixed success [3,63] is a variant of fixed effort that generates partial runs until a fixed number of them have reached the next higher level. It is illustrated in Fig. 5. We have found it to usually not be any more efficient than fixed effort, but it comes with the possibility of divergence in case the initial states of one level happen to be such that no run starting from them has the possibility to move up to the next level.

4 Scheduler sampling for nondeterminism

Resolving nondeterminism in a randomised way leads to estimates that only lie *somewhere* between the desired extremal values. In addition to computing probabilities or expected rewards, we also need to find a (near-)optimal scheduler. In our setting of undiscounted properties, this is possible using simulation-based machine learning algorithms following the ideas of [7] to incrementally improve a candidate scheduler; however, these methods cancel a key advantage of SMC: memory usage is no longer constant in the size of the state space since the scheduler's decisions for all visited states need to be stored. Currently, the only approach that does better than random resolution but keeps memory usage constant is the lightweight scheduler sampling technique of [60], which modes implements for MDP, PTA, and special classes of SA.

4.1 Lightweight scheduler sampling

The lightweight scheduler sampling (LSS) approach for MDP identifies a scheduler by a single integer (typically of 32 bits). This allows to randomly select a large number m of schedulers (i.e. integers), perform standard or rare event simulation for each, and report the maximum and minimum estimates over all sampled schedulers as approximations of the actual extremal values. We show the core of the lightweight approach—performing a simulation run for a given scheduler identifier σ—for MDP and transient properties as Algorithm 1. An MDP consists of a countable set of states S, a transition function T that maps each state to a finite *set* of probability distributions over successor states, and an initial state s_0. The algorithm uses two PRNG: $\mathscr{U}_{\text{pr}}$ to simulate the probabilistic choices (line 6), and $\mathscr{U}_{\text{nd}}$ to resolve the nondeterministic ones (line 5). We want σ to represent a deterministic memoryless scheduler: within one simulation run as well as in different runs for the same value of σ, $\mathscr{U}_{\text{nd}}$ must always make the same choice for the same state s. To achieve this, $\mathscr{U}_{\text{nd}}$ is re-initialised with a seed based on σ and s in every step (line 4). The overall effectiveness of the lightweight approach only depends on the likelihood of selecting a σ that represents a (near-)optimal scheduler. We want to sample "uniformly" from the space of all schedulers to avoid accidentally biasing against "good" schedulers. More precisely, a uniformly random choice of σ shall result in a uniformly chosen (but fixed) resolution of all nondeterministic choices. Algorithm 1 achieves this naturally for MDP.

Input: MDP $\langle S, T, s_0 \rangle$, transient property ϕ, scheduler id $\sigma \in \mathbb{Z}$

1 $s := s_0$, $\pi := s_0$
2 **while** $\phi(\pi) =$ undecided **do**
3 **if** $T(s) = \varnothing$ **then return** *false* *// deadlock: end of run*
4 $\mathscr{U}_{\text{nd}}$.initialise($\mathscr{H}(\sigma.s)$) *// use hash of σ and s to seed $\mathscr{U}_{\text{nd}}$*
5 $\mu := T(s)[\lfloor \mathscr{U}_{\text{nd}} \cdot |T(s)| \rfloor]$ *// use $\mathscr{U}_{\text{nd}}$ to select transition*
6 $s' := \mu \circ \mathscr{U}_{\text{pr}}$.next() *// use $\mathscr{U}_{\text{pr}}$ to select successor via μ*
7 $\pi := \pi.s', s := s'$ *// append s' to π and continue from s'*
8 **return** $\phi(\pi)$

Algorithm 1: MDP simulation for one scheduler id [21]

4.2 Scheduler sampling beyond MDP

LSS can be adapted to any model and type of property where the class of optimal schedulers only uses *discrete* input to make its decision for every state [46]. This is obviously the case for discrete-space discrete-time models like MDP. It means that LSS can directly be applied to MA and *time-unbounded* properties, too, since they are preserved on the MA's embedded MDP (which uses the rates only as weights to select the successor state, ignoring their semantics with respect to the passage of time). In addition to MDP and MA, modes also supports two LSS methods for PTA, based on a variant of forwards reachability with zones [21] and the region graph abstraction [46], respectively. While the former includes zone operations with worst-case runtime exponential in the number of clocks, the latter implements all operations in linear time. It exploits a novel data structure for regions based on representative valuations that performs very well in practice. Extending LSS to models with general continuous probability distributions such as stochastic automata is hindered by optimal schedulers requiring non-discrete information, namely the values and expiration times of all clocks [20]. modes currently provides prototypical LSS support for SA encoded in a particular form and various restricted classes of schedulers as described in [20,22]. We refer the interested reader to [22] for a more detailed presentation and comparison of modes' LSS methods for continuous-time models.

4.3 Scheduler histograms

The effectiveness of LSS hinges on the probability of sampling near-optimal schedulers. To allow users to investigate the distribution of schedulers, modes also returns the probabilities estimated for *all* m sampled schedulers. From this data, we can create histograms that visualise the distribution of schedulers with respect to the probabilities they induce. We show such a histogram for a PTA model of the IEEE 1394 FireWire root contention protocol (*firewire* model) in Fig. 6, and for a PTA model of IEEE 802.11 wireless LAN (*wlan* model) in Fig. 7. The properties we analyse are the probability of termination in 4000 ns for *firewire* and the probability of either of the two modelled stations' backoff counters reaching value 2 within one transmission for *wlan*. As the state spaces of both models are small enough for model checking with mcsta to be possible, we know that the minimum and maximum probabilities are, respectively, 0.781 and 1 for *firewire*, and 0.039 and 0.063 for *wlan*. We use the region-based method to perform LSS for PTA, and sample 1000 schedulers for each model. We very clearly see in the histograms that, for *firewire*, maximal schedulers are very likely to be sampled while near-minimal ones are much rarer. This indicates that the maximum probability is easily achieved while the minimum probability needs some specific scheduling choices to be made. For *wlan*, every scheduler sampled by LSS is either near-minimal or near-maximal, indicating that there are few (relevant) nondeterministic choices in this model for the property we consider.

4.4 Bounds and error accumulation

The results of an SMC analysis with LSS are lower bounds for maximum and upper bounds for minimum values up

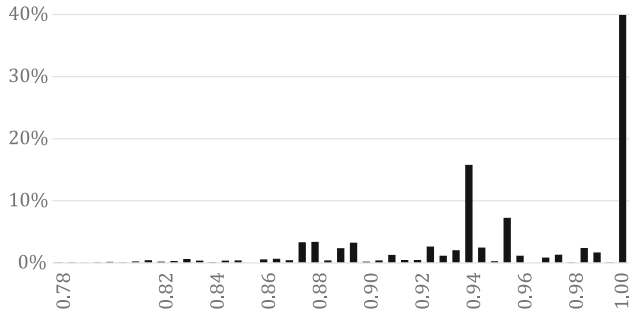

Fig. 6 Scheduler histogram for *firewire* (1000 schedulers) [22]

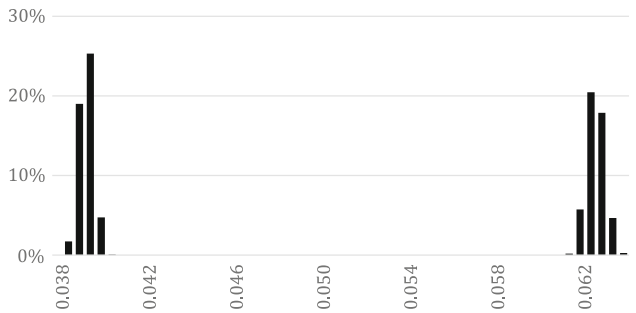

Fig. 7 Scheduler histogram for *wlan* (1000 schedulers) [22]

to the specified statistical error and confidence. They can thus be used to e.g. *disprove* safety (the maximum probability to reach an unsafe state is above a threshold) or *prove* schedulability (there is a scheduler that makes it likely to complete the workload in time), but not the opposite. The accumulation of statistical error introduced by the repeated simulation experiments over m different sampled schedulers must also be accounted for. [24] shows how to modify the Okamoto method accordingly and turn the SPRT into a correct sequential test *over schedulers*. These adjustments essentially increase the required confidence depending on the (maximum) number of schedulers to be sampled. modes also allows the Adaptive and CI method to be used with LSS by applying the standard Šidák correction for multiple comparisons to increase the required confidence: instead of the user-specified value of δ, it uses the stricter value $\delta' = \delta^{\frac{1}{m}}$. This also enables LSS for the non-Bernoulli case, i.e. for expected rewards and RES.

4.5 Two-phase and smart sampling

If an SMC analysis for fixed statistical parameters would need n runs on a deterministic model, it will need significantly more than $m \cdot n$ runs for a nondeterministic model when m schedulers are sampled due to the increase in the required confidence. modes implements a two-phase approach and smart sampling [24] to 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 generalises this principle to multiple phases, dropping only the "worst" *half* of the evaluated schedulers between phases. It starts with an informed guess of good initial values for n and m. For details, see [24]. Smart sampling tends to find more extremal schedulers faster while the two-phase approach has predictable performance as it always needs $(m+1) \cdot n$ runs. We thus use the two-phase approach for our experiments in Sect. 6.

5 Architecture and implementation

modes is implemented in C# and works on Linux, Mac OS X, and Windows systems. It builds on a solid foundation of shared infrastructure with other tools of the MODEST TOOLSET. This includes input language parsers that map MODEST, xSADF, and JANI input into a common internal metamodel for networks of stochastic hybrid automata with rewards and discrete variables. Before simulation, every model is compiled to bytecode, making the metamodel executable. The same compilation engine is used by the mcsta and prohver model checkers and the motest model-based testing tool.

We show a class diagram of the architecture of the SMC-specific part of modes in Fig. 8. Boxes represent classes, with rounded rectangles for abstract classes and invisible boxes for interfaces. Solid lines are inheritance relations. Dotted lines are associations, with double arrows for collection associations. The architecture mirrors the three distinct tasks of a statistical model checker: the generation of simulation runs and the per-run evaluation of properties, implemented in modes by *RunGenerator* and *RunEvaluator*, respectively; the coordination of simulation over multiple threads across CPU cores and networked machines, implemented by classes derived from *Worker* and *IWorkerHost*; and the statistical evaluation of simulation runs, implemented by *PropertyEvaluator*.

The central component of modes' architecture is the *Master*. It compiles the model, derives the importance function, sends both to the workers (on the same or different machines), and instantiates a *PropertiesJob* for every partition of the properties to be analysed that can share simulation runs.[2] Each *PropertiesJob* then posts simulation jobs back to the

[2] Using the same set of runs for multiple properties is an optimisation at the cost of statistical independence. modes can also be configured to generate independent runs for each property by specifying the `--independent` command-line parameter.

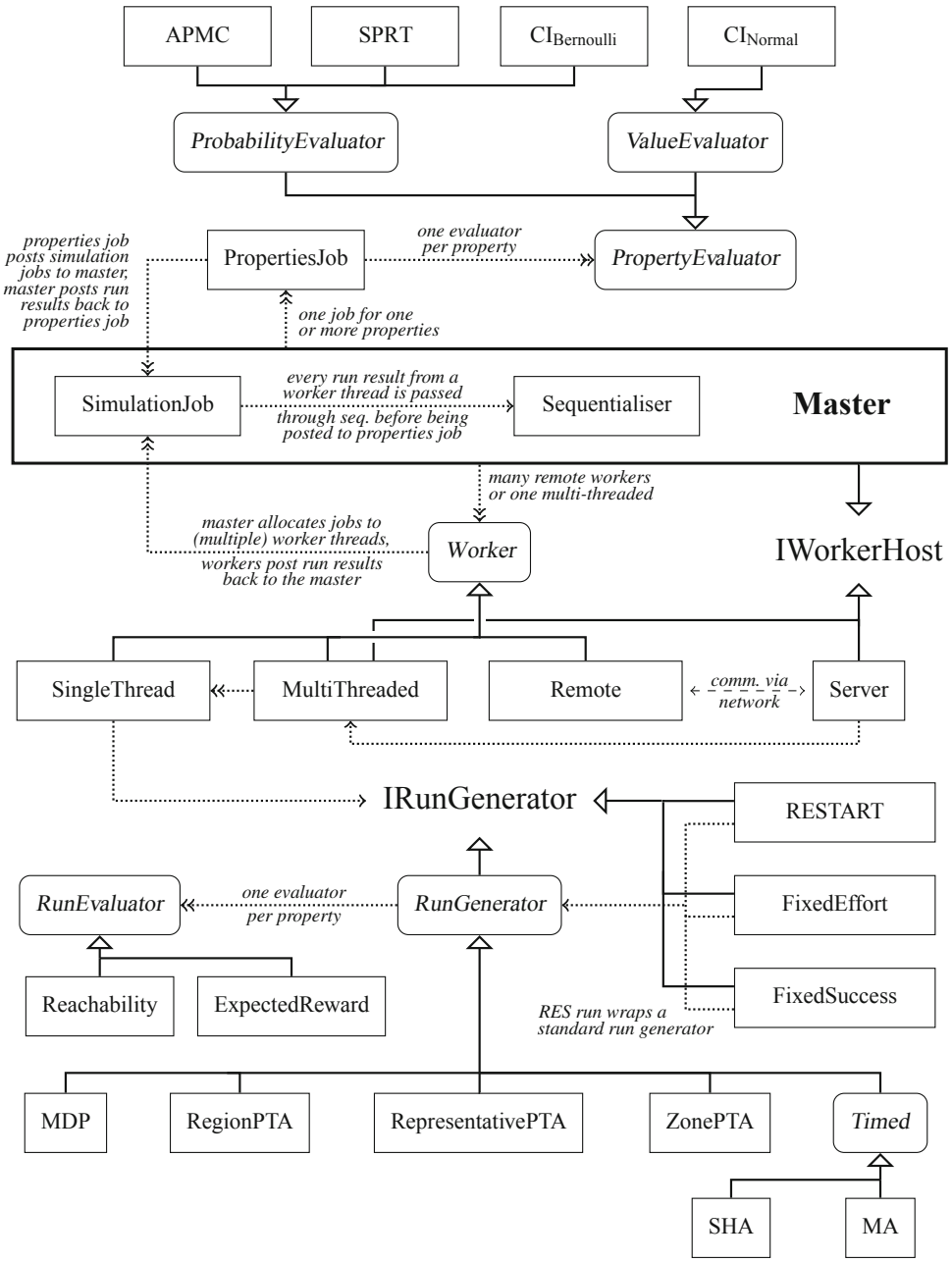

Fig. 8 The software architecture of the modes statistical model checker

master in parallel or in sequence. A simulation job is a description of how to generate and evaluate runs: which run type (i.e. *RunGenerator* derived class) to use, whether to wrap it in an importance splitting method, whether to simulate for a specific scheduler id, which compiled expressions to evaluate to determine termination and the values of the runs, etc. The master allocates posted jobs to available simulation threads offered by the workers, and notifies workers when a job is scheduled for one of their threads. As the result for an individual run is handed from the *RunEvaluator* by the *RunGenerator* via the workers to the master, it is fed into a *Sequentialiser* that implements the adaptive schedule for bias avoidance. Only after that, possibly at a later point, is it handed on to the *PropertiesJob* for statistical evaluation.

For illustration, consider a *PropertiesJob* for LSS with 10 schedulers, RES with RESTART, and the expected success method for level calculation. It is given the importance function by the master, and its first task is to compute the levels. It posts a simulation job for fixed effort runs with level information collection to the master. Depending on the current workload from other *PropertiesJob*s, the master will allocate many threads to this job. Once enough results have come in, the *PropertiesJob* terminates the simulation job, computes the levels and splitting factors, and starts with the

actual simulations: It selects 10 random scheduler identifiers and concurrently posts for each of them a simulation job for RESTART runs. The master will try to allocate available threads evenly over these jobs. As results come in, the evaluation may finish early for some schedulers, at which point the master will be instructed to stop the corresponding simulation job. It can then allocate the newly free threads to other jobs. This scheme results in a maximal exploitation of the available parallelism across workers and threads.

Due to the modularity of this architecture, it is easy to extend modes in different ways. For example, to support a new type of model (say, hybrid automata with complex nonlinear dynamics) or a new RES method, only a new class implementing *IRunGenerator* or derived from *RunGenerator* needs to be implemented. Adding another statistical evaluation method from [70] means adding a new *PropertyEvaluator*, and so on.

In distributed simulation, an instance of modes is started on each node with the `--server` parameter. This results in the creation of an instance of the *Server* class instead of a *Master*, which listens for incoming connections. Once all servers are running, a master can be started with a list of hosts to connect to. modes comes with a template script to automate this task on SLURM-based clusters.

6 Case studies

We present three case studies in this section. They have been chosen to highlight modes' capabilities in terms of the diverse types of models it supports, its ability to distribute work across compute clusters, and the new analyses possible with RES and LSS. None of them have been studied before with modes or the combinations of methods that we apply here. The first is a hybrid Petri net model with general distributions that we analyse with RES. The second is a PTA model of a low-latency wireless communication protocol to which we apply LSS. The final model is a classic RES case study of a redundant database system modelled as a CTMC, to which we add a nondeterministic repairman and investigate the impact of the repair strategy using LSS. Our experiments ran on an Intel Core i7-4790 workstation (3.6–4.0 GHz, 4 cores), a homogeneous cluster of 40 AMD Opteron 4386 nodes (3.1–3.8 GHz, 8 cores), and an inhomogeneous cluster of 15 nodes with different Intel Xeon processors. All systems run 64-bit Linux. We use 1, 2 or 4 simulation threads on the workstation (denoted "1", "2" and "4" in our tables), and n nodes with t simulation threads each on the clusters (denoted "$n \times t$"). We used a one-hour timeout, marked "—" in the tables. Note that runtimes cannot directly be compared between the workstation and the clusters due to the different architectures and configurations. We found the workstation's per-thread simulation performance to be at least twice that of the cluster, with some variation.

Data availability. The exact tool version used and the data generated in the experimental evaluation presented in this section are archived and available at DOI [40]

10.4121/uuid:64cd25f4-4192-46d1-a951-9f99b452b48f.

We use modes from version 3.0 of the MODEST TOOLSET.

6.1 Electric vehicle charging

We first consider a model of an electric vehicle charging station. It is a MODEST model adapted from the "extended" case study of [67]: a stochastic hybrid Petri net with general transitions, which in turn is based on the work in [50]. The scenario we model is of an electric vehicle being connected to the charger every evening in order to be charged the next morning. The charging process may be delayed due to high load on the power grid, and the exact time at which the vehicle is needed in the morning follows a normal distribution. We consider one week of operation and compute the probability that the desired level of charge is not reached on any $n_{fail} \in \{2, \ldots, 5\}$ of the seven mornings.

This model is not amenable to exhaustive model checking due to the non-Markovian continuous probability distributions and the hybrid dynamics modelling the charging process. However, it is deterministic (i.e. it does not contain any nondeterministic choices but is fully stochastic). We thus applied modes with standard Monte Carlo simulation (MC) as well as with RES using RESTART. We performed the same analysis on different configurations of the workstation and the homogeneous cluster. To compare MC and RES, we use CI with $\delta = 0.95$ and a relative half-with of 10 % for both. All other parameters of modes are set to default values, which implies an automatic compositional importance function and the expected success method to determine levels and splitting factors. The results are shown in Table 3. Row "conf. interval" gives the average confidence intervals that we obtained over all experiments. Note that, as explained in Sect. 2.3, these confidence intervals may not have actual confidence 0.95, but comparing the time needed to achieve the required width across the different setups is still a valid and consistent performance measure.

RES starts to noticeably pay off as soon as probabilities are in the order of 10^{-4}. The runtime of RESTART is known to depend on the levels and splitting factors, and we indeed noticed large variations in runtime for RES over several repetitions of the experiments. The runtimes for RES should thus not be used to judge the speedup with respect to parallelisation. However, when looking at the MC runtimes, we see good speedups as we increase the number of threads per node,

Table 3 Performance and scalability on the electric vehicle charging case study

	$n_{fail} = 2$		$n_{fail} = 3$		$n_{fail} = 4$		$n_{fail} = 5$	
	MC	RES	MC	RES	MC	RES	MC	RES
conf. interval	[6.4E−2, 7.8E−2]		[5.2E−3, 6.4E−3]		[2.7E−4, 3.2E−4]		[8.3E−6, 1.0E−5]	
1	2 s	4 s	30 s	19 s	585 s	206 s	—	
2	1 s	2 s	15 s	11 s	315 s	101 s		
4	1 s	1 s	8 s	5 s	163 s	69 s		
5 × 4	1 s	1 s	4 s	4 s	69 s	23 s	2241 s	496 s
5 × 8	1 s	2 s	2 s	3 s	40 s	16 s	1238 s	328 s
40 × 2	0 s	1 s	1 s	2 s	16 s	8 s	483 s	135 s
20 × 8	0 s	2 s	1 s	2 s	10 s	6 s	314 s	105 s
40 × 8	0 s	2 s	1 s	3 s	5 s	6 s	159 s	64 s

Table 4 Performance and results for the low-latency wireless network case study

		time	$\mathbb{P}(i < 4 \cup failed)$	$\mathbb{P}(i < 4 \cup offline_{\{1\}})$	$\mathbb{P}(i < 4 \cup offline_{\{2\}})$
optimal		n/a	[0.028, 0.472]	[0.026, 0.269]	[0, 0.424]
1	100	3523 s	[0.041, 0.363]	[0.030, 0.189]	[0.000, 0.309]
2	100	2045 s			
4	100	1205 s			
20 × 8	1000	607 s	[0.033, 0.383]	[0.028, 0.242]	[0.000, 0.327]
40 × 8	1000	308 s			

and near-ideal speedups as we increase the total number of nodes, as long as there is a sufficient amount of work.

Although this model was not designed with RES in mind and has only moderately rare events, the fully automated methods of modes could be applied directly, and they significantly improved performance. For a detailed experimental comparison of the RES methods implemented in modes on a larger set of examples, including events with probabilities as low as 4.8×10^{-23}, we refer the reader to [9,10].

6.2 Low-latency wireless networks

We now turn to the PTA model of a low-latency wireless networking protocol being used among three stations, originally presented in [29]. We take the original model, increase the probability of message loss and make one of the communication links nondeterministically drop messages. This allows us to study the influence of the message loss probabilities and the protocol's robustness to adversarial interference. The model *is* amenable to exhaustive model checking, as demonstrated in [29]. It allows us to show that modes can be applied to such models originally built for traditional verification, and since we can calculate the precise maximum and minimum values of all properties via model checking, we have a reference to evaluate the results of LSS.

We show the results of using modes with LSS on this model in Table 4. Row "optimal" lists the maximum and minimum probabilities computed via model checking for three properties: the probability that the protocol fails within four iterations, and that either the first or the second station goes offline. We used the two-phase LSS method with $m = 100$ schedulers on the workstation, and with $m = 1000$ schedulers on the homogeneous cluster. The intervals are the averages of the minimum and maximum values returned by all analyses. The statistical evaluation uses the Okamoto bound with $\delta = 0.95$ and $\epsilon \approx 0.0056$, which means that approx. 60,000 simulation runs are needed per scheduler.

Near-optimal schedulers for the minimum probabilities do not appear to be rare: we find good bounds for the minima even with 100 schedulers. However, for maximum probabilities, sampling more schedulers pays off in terms of better approximations. In all cases, the results are conservative approximations of the actual optima (as expected), and they are clearly more useful than the single value that would be obtained by other tools via a (hidden) randomised scheduler. Performance scales ideally with parallelism on the cluster, and still linearly on the workstation. For a deeper evaluation of the characteristics of LSS, including experiments on models too large for model checking, we refer the reader to the description of the original approach [24,60] and its extensions to PTA [21,22,46].

6.3 Redundant database system

The redundant database system [37] is a classic RES case study. It models a system consisting of six disk clusters of

Table 5 Performance and results for the reliable database system case study

	uniform scheduler			lightweight scheduler sampling (20)			
R	MC	RES	conf. interval	MC	RES	min. conf. int.	max. conf. int.
2	1 s	4 s	[1.5E−2, 1.8E−2]	4 s	31 s	[1.4E−2, 1.7E−2]	[1.5E−2, 1.9E−2]
3	8 s	3 s	[1.0E−4, 1.3E−4]	181 s	26 s	[7.9E−5, 9.6E−5]	[1.3E−4, 1.6E−4]
4	816 s	13 s	[9.3E−7, 1.1E−6]	—	221 s	[6.3E−7, 7.6E−7]	[1.3E−6, 1.6E−6]
5	—	229 s	[1.1E−8, 1.3E−8]	—	3072 s	[6.2E−9, 7.6E−9]	[1.6E−8, 2.0E−8]

$R + 2$ disks each plus two types of processors and disk controllers with R copies of each type. Component lifetimes are exponentially distributed. Components fail in one of two modes with equal probability, each mode having a different repair rate. The system is operational as long as fewer than R processors of each type, R controllers of each type, and R disks in each cluster are currently failed. The model is a CTMC with a state space too large and a transition matrix too dense for it to be amenable to model checking with symbolic tools like PRISM [57].

In the model studied in [8,10], any number of failed components can be repaired in parallel. We consider this unrealistic and extend the model by a *repairman* that can repair a single component at a time. If more than one component fails during a repair, then as soon as the current repair is finished, the repairman has to decide which to repair next. Instead of enforcing a particular repair policy as in the original model [37], we leave this decision as nondeterministic. The model thus becomes an MA. We use LSS in combination with RES to investigate the impact of the repair policy. We study the scenario where one *component* of each kind (one disk, one processor, one controller) is in failed state, and estimate the probability for *system* failure before these components are repaired. The minimum probability is achieved by a perfect repair strategy, while the maximum results from the worst possible one.

Table 5 shows the results of our LSS-plus-RES analysis with modes using default RES parameters and sampling $m = 20$ schedulers. Due to the complexity of the model, we ran this experiment on the inhomogeneous cluster only, using 16 cores on each node for 240 concurrent simulation threads in total. We see that RES needs a somewhat rare event to improve performance. We also compare LSS to the uniform randomised scheduler (as implemented in many other SMC tools). It results in a single confidence interval for the probability of failure. With LSS, we get two intervals. They do not overlap when $R \geq 3$, i.e. the repair strategy matters: a bad strategy makes failure approximately twice as likely as a good strategy! Since the results of LSS are conservative, the difference between the worst and the best strategy may be even larger.

7 Performance comparison

The case studies of the previous section highlight the abilities of modes in absolute terms. We now study its performance relative to three other SMC tools: FIG [8], PLASMA LAB [61], and PRISM's SMC engine [65]. These tools are the only current statistical model checkers we are aware of that support input languages compatible with modes: PLASMA and PRISM work with PRISM's guarded command language, which can be translated to JANI by the STORM model checker [28], while FIG uses a similar language for input–output stochastic automata (IOSA), which it can convert to JANI models of the STA model type. PLASMA and PRISM only support standard Monte Carlo simulation, and we use them to compare core simulation engine performance. FIG is a rare event simulator, and we compare its RES engine with that of modes, but using different level calculation and splitting methods.

A broader comparison of modes with probabilistic model checkers and planners—that, however, does not include other general-purpose SMC tools—has been performed as part of the QComp 2019 friendly competition. The full results of that comparison are available at qcomp.org and in the corresponding competition report [42].

Data availability. The exact version of modes used and the data generated in the experimental evaluation presented in this section, as well as scripts and instructions to replicate the experiments, are archived and available at DOI [41]

10.4121/uuid:2896b362-85d8-4705-bbe4-073fc79e23ec.

We use modes from version 3.1 of the MODEST TOOLSET, PLASMA LAB version 1.4.4, PRISM 4.5, and FIG v1.1.

7.1 Comparison with PLASMA LAB and PRISM

For the comparison of modes with PLASMA and PRISM, we use all DTMC and CTMC models from the Quantitative Verification Benchmark Set (QVBS [45]) that are available in the PRISM language, include an unbounded transient property in the case of DTMC or a time-bounded transient property in case of CTMC and have a single initial state. The latter restriction is required by all three tools. In case a model comes with multiple applicable transient properties, we arbitrarily choose one, but avoid properties for which the result is

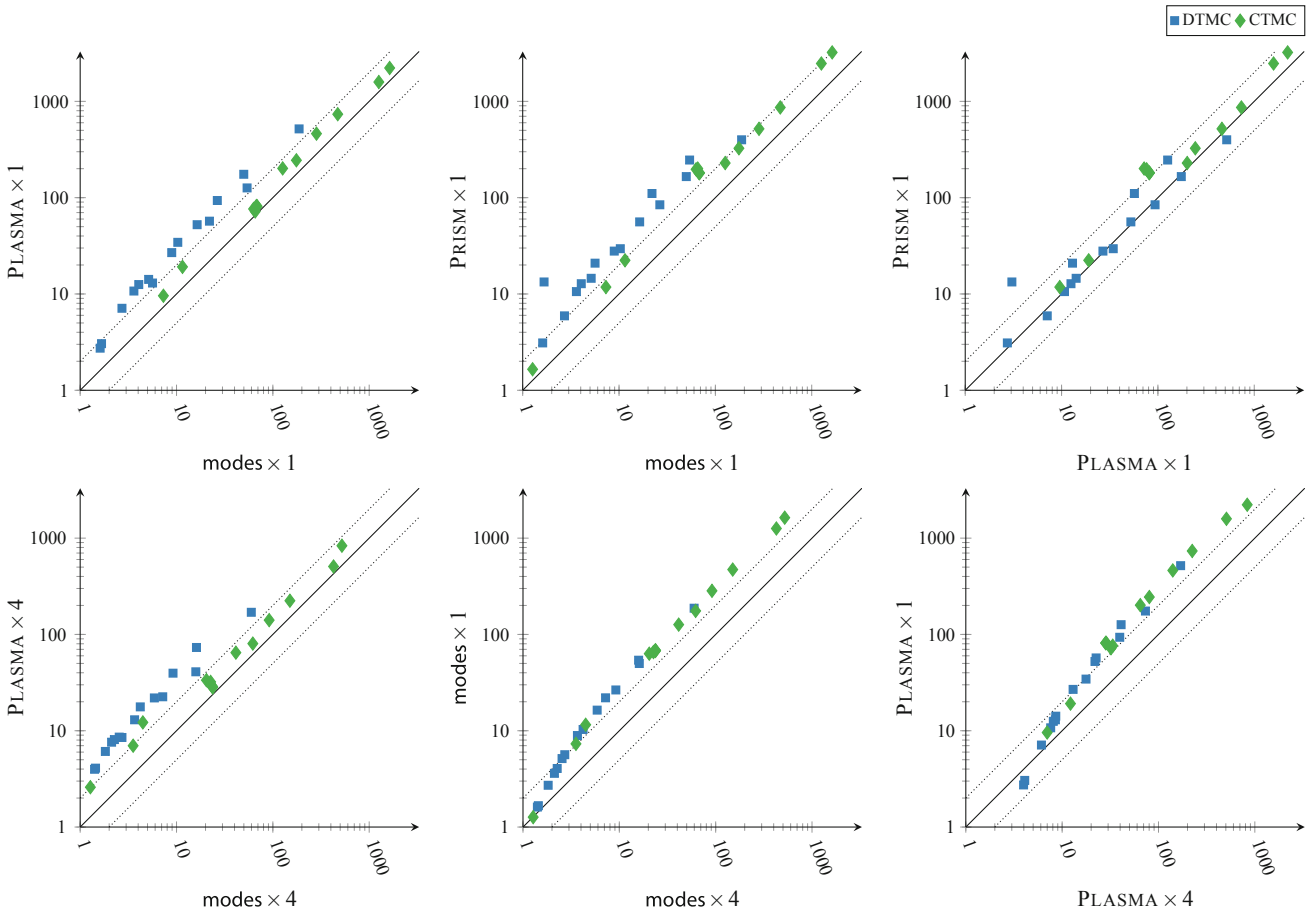

Fig. 9 Simulation times for 100,000 runs with modes, PLASMA and PRISM compared

known to be 0 or 1 where possible. Since PLASMA is restricted to bounded LTL, we turn the unbounded properties of the DTMC models into step-bounded ones by setting the step bound to a value close to, but below, the maximum simulation run length reported by PRISM for the unbounded property. All properties are queries. Most of the models have open parameters, which allows scaling their state space size. Where possible, we selected three parameter valuations from those suggested in the QVBS: one with a small, one with a medium, and one with a large state space. We observed that larger state spaces—which are not a problem per se for SMC tools due to their constant memory usage—generally lead to longer simulation runs (i.e. each run passes through more states before the property is decided) and thus longer simulation runtime. We refer to the combination of a model, a property, and a parameter valuation as a (benchmark) *instance*.

For every instance, we execute each tool five times to average out statistical fluctuations from the execution environment and different random seeds to some degree. Every execution performs $n = 100,000$ simulation runs. modes and PLASMA allow multi-threaded simulation, but PRISM does not. We thus perform every set of executions once in single-threaded mode for all three tools, and once with up to four simulation threads for modes and PLASMA. All executions are performed on the four-core workstation that we already used in Sect. 6. We checked that the values estimated by all three tools in all configurations are indeed consistent up to the statistical confidence afforded by performing 100,000 runs according to the Okamoto bound.

We visualise the results as scatter plots in Fig. 9. Each plot compares the performance of two tools. We use notation "TOOL × j" to denote tool TOOL configured to use j simulation threads. A point $\langle x, y \rangle$ in these plots represents an instance for which performing 100,000 runs took x seconds using the tool noted on the x-axis and y seconds using the tool noted on the y-axis. A point above the solid diagonal line thus indicates an instance where the x-axis tool was faster; a point above (below) the upper (lower) dotted line indicates an instance where the x-axis tool was more than twice as fast as the y-axis tool.

We see (in the left two plots of the first row and in the leftmost plot of the second row) that modes clearly out-

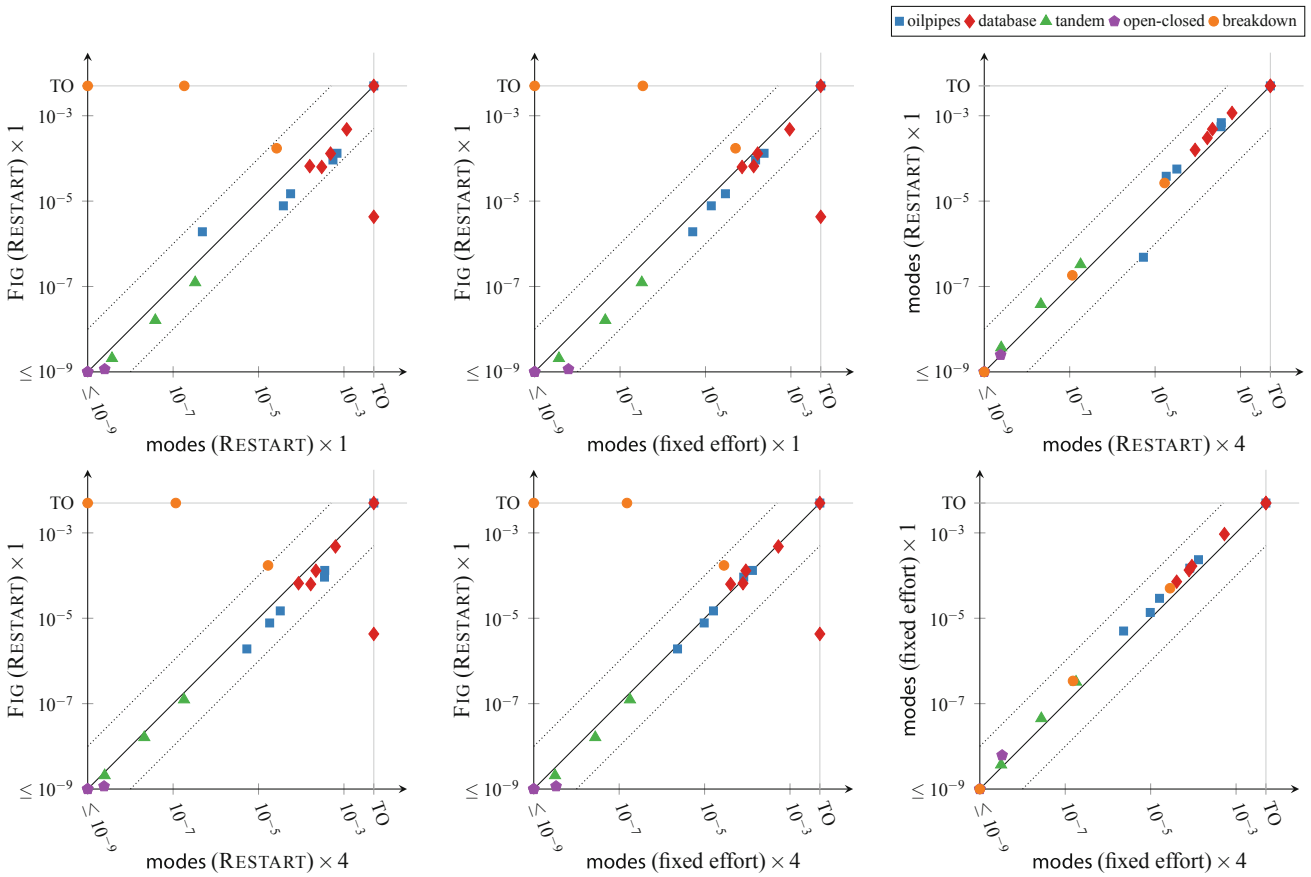

Fig. 10 Confidence interval widths after 10 min of simulation time with FIG and modes compared

performs both PLASMA and PRISM on all DTMC instances. Here, modes uses the simulation algorithm outlined in Sect. 2.1.1. In single-threaded mode, PLASMA takes rather consistently between 2.3 and 3.5 times as long to perform the same number of runs, with an average factor of 2.8. The only outlier is the `leader_sync` model, which is very fast to simulate, where PLASMA only needs about 75 % more time. PRISM needs on average 3.5 times as long as modes, with significant outliers. With four simulation threads, the difference between modes and PLASMA increases, and PLASMA now takes on average 3.5 times as long as modes. When it comes to CTMC, where modes uses the algorithm of Sect. 2.1.2, it is still faster than PLASMA and PRISM, albeit not as much as for DTMC (the average runtime factors being only 1.3 for PLASMA, and 2.2 for PRISM).

For all DTMC and most CTMC models, modes realises a higher speedup from multi-threaded simulation than PLASMA. Using four threads, it performs on average 2.3 times as many DTMC simulation runs as using one thread in the same time, and 2.7 times as many CTMC runs: the more involved CTMC simulation algorithm scales better with the number of threads, which was expected since it can perform more independent work before it has to synchronise results back to the main thread. The corresponding factors for PLASMA are 1.9 and 2.5, respectively. That the factors are all noticeably less than four was expected since the workstation only has four physical cores, which must also run the simulation management and result aggregation code as well as background tasks.

7.2 Comparison with FIG

To compare modes with FIG, we use the benchmark set on which FIG was originally evaluated [8], consisting of the *breakdown*, *database*, *oilpipes*, *open-closed*, and *tandem* IOSA models. The *breakdown*, *open-closed*, and *tandem* models represent queueing systems with exponentially distributed delays that are frequently used RES benchmarks. The *database* and *oilpipes* models capture highly reliable systems and use exponential, log-normal, and Rayleigh distributions. We use an unbounded transient query for each of them and again include several instances of each model with different parameter valuations. FIG runs directly on the IOSA models, while modes uses JANI translations generated by FIG. These translations map to the general

STA model type in JANI, which is supported in modes by the SHA simulation algorithm outlined in Sect. 2.1.4. Since the properties capture rare events, we use both tools' rare event simulation engines. FIG thus uses the sequential Monte Carlo method to select levels, and RESTART simulation runs; we run modes with the default expected success method to select levels and splitting factors and use both RESTART as well as fixed effort simulation runs. Our evaluation thus compares not only the core (rare event) simulation implementation performance of the two tools, but also the behaviour of the level selection and splitting methods.

As in the DTMC/CTMC comparison, we execute each tool five times per instance and use the same workstation machine. FIG does not support multi-threaded simulation; we compare it to modes using one and four threads to still be able to quantify the advantage gained by implementing multi-threaded simulation. Every tool is executed on every instance for 10 min; at that point, the execution is stopped, and the tool reports the confidence interval obtained from the runs completed so far. The performance measure is thus the (absolute) width of the confidence interval, with a narrower interval indicating better performance. Note that the number of runs necessary for a certain interval width depends roughly quadratically on the width.

We again visualise the results as scatter plots, shown in Fig. 10. Points $\langle x, y \rangle$ now represent absolute confidence interval widths x and y. The "TO" lines indicate cases where, in at least one of the five executions, the corresponding tool did not manage to produce a nonzero estimate in 10 min, either because the importance function and level computation took too long, or because it never encountered the rare event at all. The dotted diagonal lines delineate a factor of 10 difference in terms of confidence interval width. We see that FIG manages to perform RES more efficiently for all models except for the *breakdown* queueing system. modes visibly gains from its support of multi-threaded simulation. Notably, the fixed effort splitting method with four threads manages to be more efficient than FIG on some instances of the *oilpipes* and *database* models, too. Overall, we see that FIG's combination of level calculation and splitting methods together with its specialised simulation algorithm for IOSA remains the tool of choice for IOSA models, yet we find modes' performance using a general-purpose simulation algorithm for SHA competitive as well.

8 Conclusion

We presented modes, the MODEST TOOLSET's distributed statistical model checker. It provides methods to tackle both of the prominent challenges in simulation: nondeterminism and rare events. Its modular software architecture allows its various features to be easily combined and extended. We used lightweight scheduler sampling with Markov automata and combined it with rare event simulation to gain insights into a challenging case study that, currently, cannot be analysed for the same aspects with any other tool that we are aware of. We have shown that the simulation performance of modes is comparable with or better than that of other SMC tools working with similar models. modes is available for download as part of the MODEST TOOLSET at

`www.modestchecker.net`.

Acknowledgements The authors thank Cyrille Jégourel (Singapore University of Technology and Design) for helpful discussions on statistical evaluation methods (Sect. 2.3), Carina Pilch (Westfälische Wilhelms-Universität Münster) for her support in understanding the original model of the *vehicle charging* case study (Sect. 6.1), and Sebastian Junges (RWTH Aachen) for implementing and supporting the *wireless networks* case study model (Sect. 6.2).

References

1. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Trans. Model. Comput. Simul. **28**(1), 6:1–6:39 (2018). https://doi.org/10.1145/3158668
2. Agresti, A., Coull, B.A.: Approximate is better than "exact" for interval estimation of binomial proportions. Am. Stat. **52**(2), 119–126 (1998). https://doi.org/10.2307/2685469
3. Amrein, M., Künsch, H.R.: A variant of importance splitting for rare event estimation: fixed number of successes. ACM Trans. Model. Comput. Simul. **21**(2), 13:1–13:20 (2011). https://doi.org/10.1145/1899396.1899401
4. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
5. Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and statistical model checking for Modestly nondeterministic models. In: MMB/DFT, LNCS, vol. 7201, pp. 249–252. Springer (2012). https://doi.org/10.1007/978-3-642-28540-0_20
6. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.: MoDeST: a compositional modeling formalism for hard and softly timed systems. IEEE Trans. Softw. Eng. **32**(10), 812–830 (2006). https://doi.org/10.1109/TSE.2006.104
7. 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, LNCS,

vol. 8837, pp. 98–114. Springer (2014). https://doi.org/10.1007/978-3-319-11936-6_8
8. Budde, C.E.: Automation of importance splitting techniques for rare event simulation. Ph.D. thesis, Universidad Nacional de Córdoba, Córdoba, Argentina (2017)
9. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Better automated importance splitting for transient rare events. In: SETTA, LNCS, vol. 10606, pp. 42–58. Springer (2017). https://doi.org/10.1007/978-3-319-69483-2_3
10. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Automated compositional importance splitting. Sci. Comput. Program. **174**, 90–108 (2019). https://doi.org/10.1016/j.scico.2019.01.006
11. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: A statistical model checker for nondeterminism and rare events. In: TACAS, LNCS, vol. 10806, pp. 340–358. Springer (2018). https://doi.org/10.1007/978-3-319-89963-3_20
12. Budde, C.E., D'Argenio, P.R., Monti, R.E.: Compositional construction of importance functions in fully automated importance splitting. In: VALUETOOLS. ICST (2016). https://doi.org/10.4108/eai.25-10-2016.2266501
13. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: TACAS, LNCS, vol. 10206, pp. 151–168. Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_9
14. Bulychev, P.E., David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Checking and distributing statistical model checking. In: NASA Formal Methods, LNCS, vol. 7226, pp. 449–463. Springer (2012). https://doi.org/10.1007/978-3-642-28891-3_39
15. Cérou, F., Del Moral, P., Furon, T., Guyader, A.: Sequential Monte Carlo for rare event estimation. Stat. Comput. **22**(3), 795–808 (2012). https://doi.org/10.1007/s11222-011-9231-6
16. Chen, J., Xu, J.: Sampling adaptively using the Massart inequality for scalable learning. In: ICMLA, pp. 362–367. IEEE (2013). https://doi.org/10.1109/ICMLA.2013.149
17. Chow, Y.S., Robbins, H.: On the asymptotic theory of fixed-width sequential confidence intervals for the mean. Ann. Math. Stat. **36**(2), 457–462 (1965). https://doi.org/10.1214/aoms/1177700156
18. Clopper, C.J., Pearson, E.S.: The use of confidence or fiducial limits illustrated in the case of the binomial. Biometrika **26**(4), 404–413 (1934). https://doi.org/10.2307/2331986
19. Daca, P., Henzinger, T.A., Kretínský, J., Petrov, T.: Faster statistical model checking for unbounded temporal properties. ACM Trans. Comput. Log. **18**(2), 12:1–12:25 (2017). https://doi.org/10.1145/3060139
20. D'Argenio, P.R., Gerhold, M., Hartmanns, A., Sedwards, S.: A hierarchy of scheduler classes for stochastic automata. In: FoSSaCS, LNCS, vol. 10803. Springer (2018). https://doi.org/10.1007/978-3-319-89366-2_21
21. D'Argenio, P.R., Hartmanns, A., Legay, A., Sedwards, S.: Statistical approximation of optimal schedulers for probabilistic timed automata. In: iFM, LNCS, vol. 9681, pp. 99–114. Springer (2016). https://doi.org/10.1007/978-3-319-33693-0_7
22. D'Argenio, P.R., Hartmanns, A., Sedwards, S.: Lightweight statistical model checking in nondeterministic continuous time. In: ISoLA, LNCS, vol. 11245, pp. 336–353. Springer (2018). https://doi.org/10.1007/978-3-030-03421-4_22
23. D'Argenio, P.R., Katoen, J.P.: A theory of stochastic systems part I: stochastic automata. Inf. Comput. **203**(1), 1–38 (2005). https://doi.org/10.1016/j.ic.2005.07.001
24. D'Argenio, P.R., Legay, A., Sedwards, S., Traonouez, L.M.: Smart sampling for lightweight verification of Markov decision processes. STTT **17**(4), 469–484 (2015). https://doi.org/10.1007/s10009-015-0383-0
25. David, A., Jensen, P.G., Larsen, K.G., Mikucionis, M., Taankvist, J.H.: Uppaal stratego. In: TACAS, LNCS, vol. 9035, pp. 206–211. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_16
26. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: CAV, LNCS, vol. 6806, pp. 349–355. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_27
27. Dean, T., Dupuis, P.: Splitting for rare event simulation: a large deviation approach to design and analysis. Stoch. Process. Appl. **119**(2), 562–587 (2009). https://doi.org/10.1016/j.spa.2008.02.017
28. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A Storm is coming: A modern probabilistic model checker. In: CAV, LNCS, vol. 10427, pp. 592–600. Springer (2017). https://doi.org/10.1007/978-3-319-63390-9_31
29. Dombrowski, C., Junges, S., Katoen, J.P., Gross, J.: Model-checking assisted protocol design for ultra-reliable low-latency wireless networks. In: SRDS, pp. 307–316. IEEE (2016). https://doi.org/10.1109/SRDS.2016.048
30. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS, pp. 342–351. IEEE Computer Society (2010). https://doi.org/10.1109/LICS.2010.41
31. Forejt, V., Kwiatkowska, M.Z., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: SFM, LNCS, vol. 6659, pp. 53–113. Springer (2011). https://doi.org/10.1007/978-3-642-21455-4_3
32. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and safety verification for stochastic hybrid systems. In: HSCC, pp. 43–52. ACM (2011). https://doi.org/10.1145/1967701.1967710
33. Frey, J.: Fixed-width sequential confidence intervals for a proportion. Am. Stat. **64**(3), 242–249 (2010). https://doi.org/10.1198/tast.2010.09140
34. Garvels, M.J.J.: The splitting method in rare event simulation. Ph.D. thesis, University of Twente, Enschede, The Netherlands (2000)
35. Garvels, M.J.J., Kroese, D.P.: A comparison of RESTART implementations. In: Winter Simulation Conference, pp. 601–608 (1998). https://doi.org/10.1109/WSC.1998.745040
36. Garvels, M.J.J., van Ommeren, J.C.W., Kroese, D.P.: On the importance function in splitting simulation. Eur. Trans. Telecommun. **13**(4), 363–371 (2002). https://doi.org/10.1002/ett.4460130408
37. Goyal, A., Shahabuddin, P., Heidelberger, P., Nicola, V.F., Glynn, P.W.: A unified framework for simulating Markovian models of highly dependable systems. IEEE Trans. Comput. **41**(1), 36–51 (1992). https://doi.org/10.1109/12.123381
38. Graf-Brill, A., Hartmanns, A., Hermanns, H., Rose, S.: Modelling and certification for electric mobility. In: Industrial Informatics (INDIN). IEEE (2017). https://doi.org/10.1109/INDIN.2017.8104755
39. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods Syst. Des. **43**(2), 191–232 (2013). https://doi.org/10.1007/s10703-012-0167-z
40. Hartmanns, A.: A statistical model checker for nondeterminism and rare events (artifact). 4TU. Centre for Research Data (2018). https://doi.org/10.4121/uuid:64cd25f4-4192-46d1-a951-9f99b452b48f
41. Hartmanns, A.: An efficient statistical model checker for nondeterminism and rare events (artifact). 4TU.Centre for Research Data (2019). https://doi.org/10.4121/uuid:2896b362-85d8-4705-bbe4-073fc79e23ec
42. Hartmanns, A., Hensel, C., Klauck, M., Klein, J., Kretínský, J., Parker, D., Quatmann, T., Ruijters, E., Steinmetz, M.: The 2019 comparison of tools for the analysis of quantitative formal models. In: TACAS, LNCS, vol. 11429. Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_5
43. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In:

TACAS, LNCS, vol. 8413, pp. 593–598. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_51
44. Hartmanns, A., Hermanns, H., Bungert, M.: Flexible support for time and costs in scenario-aware dataflow. In: EMSOFT. ACM (2016). https://doi.org/10.1145/2968478.2968496
45. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: TACAS, LNCS, vol. 11427. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_20
46. Hartmanns, A., Sedwards, S., D'Argenio, P.R.: Efficient simulation-based verification of probabilistic timed automata. In: Winter Simulation Conference (2017). https://doi.org/10.1109/WSC.2017.8247885
47. Hartmanns, A., Timmer, M.: Sound statistical model checking for MDP using partial order and confluence reduction. STTT **17**(4), 429–456 (2015). https://doi.org/10.1007/s10009-014-0349-7
48. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**(1), 94–124 (1998). https://doi.org/10.1006/jcss.1998.1581
49. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI, LNCS, vol. 2937, pp. 73–84. Springer (2004). https://doi.org/10.1007/978-3-540-24622-0_8
50. Hüls, J., Remke, A.: Coordinated charging strategies for plug-in electric vehicles to ensure a robust charging process. In: VALUETOOLS. ICST (2016). https://doi.org/10.4108/eai.25-10-2016.2266997
51. Jégourel, C., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Importance sampling for stochastic timed automata. In: SETTA, LNCS, vol. 9984, pp. 163–178. Springer (2016). https://doi.org/10.1007/978-3-319-47677-3_11
52. Jégourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: CAV, LNCS, vol. 8044, pp. 576–591. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_38
53. Jégourel, C., Legay, A., Sedwards, S.: Command-based importance sampling for statistical model checking. Theor. Comput. Sci. **649**, 1–24 (2016). https://doi.org/10.1016/j.tcs.2016.08.009
54. Jégourel, C., Legay, A., Sedwards, S., Traonouez, L.M.: Distributed verification of rare properties using importance splitting observers. ECEASST **72** (2015). https://doi.org/10.14279/tuj.eceasst.72.1024
55. Kearns, M.J., Mansour, Y., Ng, A.Y.: A sparse sampling algorithm for near-optimal planning in large Markov decision processes. Mach. Learn. **49**(2–3), 193–208 (2002). https://doi.org/10.1023/A:1017932429737
56. Kurkowski, S., Camp, T., Colagrosso, M.: MANET simulation studies: the incredibles. Mobile Comput. Commun. Rev. **9**(4), 50–61 (2005). https://doi.org/10.1145/1096166.1096174
57. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV, LNCS, vol. 6806, pp. 585–591. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_47
58. 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). https://doi.org/10.1016/S0304-3975(01)00046-9
59. L'Ecuyer, P., Le Gland, F., Lezaud, P., Tuffin, B.: Splitting techniques. In: Rubino, G., Tuffin, B. (eds.) Rare Event Simulation Using Monte Carlo Methods, pp. 39–61. Wiley (2009). https://doi.org/10.1002/9780470745403.ch3
60. Legay, A., Sedwards, S., Traonouez, L.: Scalable verification of Markov decision processes. In: WS-FMDS at SEFM, LNCS, vol. 8938, pp. 350–362. Springer (2014). https://doi.org/10.1007/978-3-319-15201-1_23
61. Legay, A., Sedwards, S., Traonouez, L.M.: Plasma Lab: a modular statistical model checking platform. ISoLA, LNCS **9952**, 77–93 (2016). https://doi.org/10.1007/978-3-319-47166-2_6
62. Legay, A., Sedwards, S., Traonouez, L.M.: Rare events for statistical model checking an overview. In: Reachability Problems, LNCS, vol. 9899, pp. 23–35. Springer (2016). https://doi.org/10.1007/978-3-319-45994-3_2
63. LeGland, F., Oudjane, N.: A sequential particle algorithm that keeps the particle system alive, pp. 351–389. Springer (2006). https://doi.org/10.1007/11587392_11
64. Matsumoto, M., Nishimura, T.: Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**(1), 3–30 (1998). https://doi.org/10.1145/272991.272995
65. Nimal, V.: Statistical approaches for probabilistic model checking. Master's thesis, Oxford University (2010)
66. Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. Ann. Inst. Stat. Math. **10**(1), 29–35 (1959). https://doi.org/10.1007/BF02883985
67. Pilch, C., Remke, A.: Statistical model checking for hybrid Petri nets with multiple general transitions. In: DSN, pp. 475–486. IEEE Computer Society (2017). https://doi.org/10.1109/DSN.2017.41
68. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. Wiley, New York (1994)
69. Reijsbergen, D., de Boer, P., Scheinhardt, W.R.W.: Hypothesis testing for rare-event simulation: Limitations and possibilities. In: ISoLA, LNCS, vol. 9952, pp. 16–26. Springer (2016). https://doi.org/10.1007/978-3-319-47166-2_2
70. Reijsbergen, D., de Boer, P., Scheinhardt, W.R.W., Haverkort, B.R.: On hypothesis testing for statistical model checking. STTT **17**(4), 377–395 (2015). https://doi.org/10.1007/s10009-014-0350-1
71. Rolland, J., Simonnet, E.: Statistical behaviour of adaptive multilevel splitting algorithms in simple models. J. Comput. Phys. **283**, 541–558 (2015). https://doi.org/10.1016/j.jcp.2014.12.009
72. Rubino, G., Tuffin, B.: Introduction to Rare Event Simulation. In: In: Rubino, G., Tuffin, B. (eds.) Rare Event Simulation Using Monte Carlo Methods, pp. 1–13. Wiley (2009). https://doi.org/10.1002/9780470745403.ch1
73. Thulin, M.: The cost of using exact confidence intervals for a binomial proportion. Electron. J. Stat. **8**(1), 817–840 (2014). https://doi.org/10.1214/14-EJS909
74. Turati, P., Pedroni, N., Zio, E.: Advanced RESTART method for the estimation of the probability of failure of highly reliable hybrid dynamic systems. Reliab. Eng. Syst. Safety **154**(C), 117–126 (2016). https://doi.org/10.1016/j.ress.2016.04.020
75. Villén-Altamirano, J.: RESTART vs splitting: a comparative study. Perform. Eval. **121–122**, 38–47 (2018). https://doi.org/10.1016/j.peva.2018.02.002
76. Villén-Altamirano, M., Villén-Altamirano, J.: RESTART: a method for accelerating rare event simulations. In: Queueing, Performance and Control in ATM (ITC-13), pp. 71–76. Elsevier (1991)
77. Villén-Altamirano, M., Villén-Altamirano, J.: The rare event simulation method RESTART: efficiency analysis and guidelines for its application. In: Network Performance Engineering, LNCS, vol. 5233, pp. 509–547. Springer (2011). https://doi.org/10.1007/978-3-642-02742-0_22
78. Wald, A.: Sequential tests of statistical hypotheses. Ann. Math. Stat. **16**(2), 117–186 (1945). https://doi.org/10.1214/aoms/1177731118
79. Wald, A., Wolfowitz, J.: Optimum character of the sequential probability ratio test. Ann. Math. Stat. **19**(3), 326–339 (1948). https://doi.org/10.1214/aoms/1177730197

80. Younes, H.L.S.: Ymer: A statistical model checker. In: CAV, *LNCS*, vol. 3576, pp. 429–433. Springer (2005). https://doi.org/10.1007/11513988_43
81. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: CAV, vol. 2404, pp. 223–235. Springer (2002). https://doi.org/10.1007/3-540-45657-0_17
82. Zimmermann, A., Lavista, A.C., Rodríguez, R.J.: Some notes on rare-event simulation challenges. In: VALUETOOLS, pp. 263–264. ACM (2017). https://doi.org/10.1145/3150928.3150963