Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Smart Sampling for Lightweight Verification of Markov Decision Processes

Pedro D'Argenio[1], Axel Legay[2], Sean Sedwards[2], Louis-Marie Traonouez[2]

[1] Universidad Nacional de Córdoba, Argentina
[2] Inria Rennes – Bretagne Atlantique, France

The date of receipt and acceptance will be inserted by the editor

Abstract. Markov decision processes (MDP) are useful to model optimisation problems in concurrent systems. To verify MDPs with efficient Monte Carlo techniques requires that their nondeterminism be resolved by a scheduler. Recent work has introduced the elements of lightweight techniques to sample directly from scheduler space, but finding optimal schedulers by simple sampling may be inefficient. Here we describe "smart" sampling algorithms that can make substantial improvements in performance.

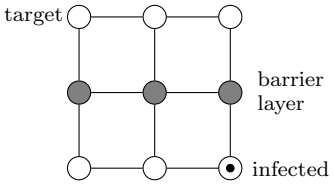


Fig. 1: Model of network virus infection.

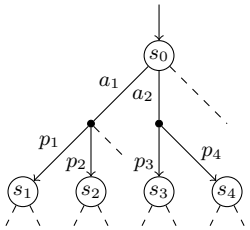


Fig. 2: Fragment of a Markov decision process.

1 Introduction

Markov decision processes describe systems that interleave nondeterministic *actions* and probabilistic transitions. This model has proved useful in many real optimisation problems [33–35] and may be used to represent concurrent probabilistic programs (see, e.g., [3,1]). Such models comprise probabilistic subsystems whose transitions depend on the states of the other subsystems, while the order in which concurrently enabled transitions execute is nondeterministic. This order may radically affect the behaviour of a system and it is thus useful to calculate the upper and lower bounds of quantitative aspects of performance.

As an example, consider the network of computational nodes depicted in Fig. 1 (relating to the case study in Section 6.4). Given that one of the nodes is infected by a virus, we would like to calculate the probability that a target node becomes infected. If we know the probability that the virus will pass from one node to the next, we could model the system as a discrete time Markov chain and analyse it to find the probability that any particular node will become infected. Such a model ignores the possibility that the virus might actually choose which node to infect, e.g., to maximise its probability of passing through the barrier layer. Under such circumstances some nodes might be infected with near certainty or with only very low probability, but this would not be adequately captured by the Markov chain. By modelling the virus's choice of node as a nondeterministic transition in an MDP, the maximum and minimum probabilities of infection can be considered.

Figure 2 shows a typical fragment of an MDP. In a given state (s_0), an action ($a_1, a_2, \dots$) is chosen nondeterministically to select a distribution of probabilistic transitions ($p_1, p_2, \dots$ or p_3, p_4, etc.). A probabilistic choice is then made to select the next state ($s_1, s_2, s_3, s_4, \dots$). In this work we use the term *scheduler* to refer to a particular way the nondeterminism in an MDP is resolved. We consider memoryless schedulers, whose choices depend only on the current state, and history-dependent schedulers whose choices may also depend on previous states.

Classic analysis of MDPs is concerned with finding the expected maximum or minimum reward for an execution of the system, given individual rewards assigned to each of the actions [2,31]. Rewards may also be assigned to states or transitions between states [21]. Here

we focus on MDPs in the context of model checking concurrent probabilistic systems, to find schedulers that maximise or minimise the probability of a property. Model checking is an automatic technique to verify that a system satisfies a property specified in temporal logic [7]. *Probabilistic* model checking quantifies the probability that a probabilistic system will satisfy a property [13]. Numerical model checking algorithms to solve purely probabilistic systems are costly in time and space. Finding extremal probabilities in MDPs is generally more so, but is nevertheless a polynomial function of the explicit description of the MDP [3].

Statistical model checking (SMC) describes a collection of Monte Carlo sampling techniques that make probabilistic model checking more tractable by returning approximative results with statistical confidence [37]. SMC algorithms generally avoid constructing an explicit representation of the state space of a system, employing a compact executable model to generate states on the fly during simulation. SMC is therefore efficient for large, possibly infinite state, systems. Moreover, since the simulations are required to be statistically independent, SMC may be efficiently divided on parallel computing architectures. Recent approaches to apply SMC to MDPs are memory-intensive [4,15,26,14,6] or do not find schedulers that optimise probabilities [4,26,14]. Classic sampling approaches for MDPs, such as the Kearns algorithm [19], are memory-efficient but address a different problem related to discounted MDPs.

This work extends [27]. In [27] the authors provide sampling techniques that can form the basis of memory-efficient ("lightweight") verification of MDPs. The principal contributions of [27] are (*i*) specifying the infinite behaviour of schedulers using $\mathcal{O}(1)$ memory, (*ii*) sampling directly and uniformly from scheduler space, and (*iii*) quantifying the statistical confidence of multiple estimates or multiple hypothesis tests. As in the case of standard SMC, sampling makes the verification problem independent of the size of the space of samples, with a convergence to the correct result almost surely guaranteed with an infinite number of samples. The use of lightweight techniques opens up the possibility to efficiently distribute the problem on high performance massively parallel architectures, such as general purpose computing on graphics processing units (GPGPU).

Sampling schedulers makes a significant advance over mere enumeration. For example, suppose half of all schedulers for a given MDP and property are "near optimal", i.e., have a probability of satisfying the property that is deemed adequately close to the true optimum. If all such near optimal schedulers lie in the second half of the enumeration, it will be necessary to enumerate half of all schedulers before finding one that is near optimal. In contrast, one would expect to see a near optimal scheduler after just two random selections, i.e., the expectation with two samples is one. This phenomenon is not limited to the case when schedulers are pathologically distributed with respect to the enumeration. Since the total number of schedulers increases exponentially with path length, it is usually very large. Hence, even when near optimal schedulers are more uniformly distributed with respect to the enumeration, it is typically not tractable to use enumeration to find one. Note that sampling also works with non-denumerable spaces. The cost of finding a near optimal scheduler with sampling is simply proportional to the relative mass of near optimal schedulers in scheduler space. Our experiments with standard case studies suggest that this cost is often reasonable.

It was demonstrated in [27] that simple undirected sampling may be adequate for some case studies. In this work we present "smart sampling" algorithms that make significantly better use of a simulation budget. For a given number of candidate schedulers, smart sampling can reduce the simulation cost of extremal probability estimation by more than $N/\lceil 2+\log_2 N\rceil$, where N is the minimum number of simulations necessary to achieve the required statistical confidence, as given by (3). The basic notions of smart sampling were hinted at in [27]. Simply put, a small part of the budget is used to perform an initial assessment of the problem and to generate an optimal initial candidate set of schedulers. The remaining budget is used to test and refine the candidate set: suboptimal schedulers are removed and their budget is reallocated to good ones. Here we give a full exposition of smart sampling and explain its limitations. We have implemented the algorithms in our statistical model checking platform, PLASMA[1], and demonstrate their successful application to a number of case studies from the literature. We include some examples that are intractable to numerical techniques and compare the performance of our techniques with an alternative sampling approach [15]. We also give an example where smart sampling is less effective, but show that the results may nevertheless be useful in bounding the range of extremal probabilities.

Structure of the Paper

In Section 2 we briefly survey closely related work. In Section 3 we introduce some definitions and notation necessary for the sequel. In Sections 4 we recall the basis of our lightweight verification techniques. In Section 5 we describe the notion of smart sampling and present our smart estimation and smart hypothesis testing algorithms. In Section 6 we give the results of experiments with a number of case studies from the literature. In Section 7 we discuss the limitations of smart sampling and in Section 8 we summarise the challenges and prospects for our approach.

[1] project.inria.fr/plasma-lab/

2 Related Work

The classic algorithms to solve MDPs are 'policy iteration' and 'value iteration' [31]. Model checking algorithms for MDPs may use value iteration applied to probabilities [1, Ch. 10] or solve the same problem using linear programming [3]. The principal challenge of finding optimal schedulers is what has been described as the 'curse of dimensionality' [2] and the 'state explosion problem' [7]. In essence, these two terms refer to the fact that the number of states of a system increases exponentially with respect to the number of interacting components and state variables. This phenomenon has motivated the design of lightweight sampling algorithms that find 'near optimal' schedulers to optimise rewards in discounted MDPs [19], but the standard model checking problem of finding extremal probabilities in non-discounted MDPs is significantly more challenging. Since nondeterministic and probabilistic choices are interleaved in an MDP, schedulers are typically of the same order of complexity as the system as a whole and may be infinite. As a result, previous SMC algorithms for MDPs have considered only memoryless schedulers or have other limitations.

The Kearns algorithm [19] is the classic 'sparse sampling algorithm' for large, infinite horizon, discounted MDPs. It constructs a 'near optimal' scheduler by approximating the best action from a current state, using a stochastic depth-first search. Importantly, optimality is with respect to discounted rewards, not probability. The algorithm can work with large, potentially infinite state MDPs because it explores a probabilistically bounded search space. This, however, is exponential in the discount. To find the action with the greatest expected reward in the current state of a trace, the algorithm recursively estimates the rewards of successor states, up to some maximum depth implicitly defined by the discount and an error threshold. Actions are enumerated while probabilistic choices are explored by sampling, with the number of samples set as a parameter. The discount guarantees that the algorithm eventually converges. The stopping criterion is when successive estimates differ by less than the error threshold. Since the actions of a state are re-evaluated every time the state is visited (because actions are history-dependent), the performance of the Kearns algorithm is critically dependent on its parameters.

There have been several recent attempts to apply SMC to nondeterministic models [4,26,15,14,27,6].

In [4,14] the authors present on-the-fly algorithms to remove 'spurious' nondeterminism, so that standard SMC may be used. This approach is limited to the class of models whose nondeterminism does not affect the resulting probability of a property. The algorithms therefore do not attempt to address the standard MDP model checking problems related to finding optimal schedulers.

In [26] the authors first find a memoryless scheduler that is near optimal with respect to a discounted reward scheme, using an adaptation of the Kearns algorithm. This induces a Markov chain whose properties may be verified with standard SMC. By storing and re-using the choices in visited states, the algorithm improves on the performance of the Kearns algorithm, but is thus limited to tractable memoryless schedulers. The near optimality of the induced Markov chain is with respect to discounted rewards, not probability, hence [26] does not address the standard model checking problems of MDPs.

In [15] the authors present an SMC algorithm to decide whether there exists a memoryless scheduler for a given MDP, such that the probability of a property is above a specified threshold. The algorithm has an inner loop that generates candidate schedulers by iteratively improving a probabilistic scheduler, according to sample traces that satisfy the property. The algorithm is limited to memoryless schedulers because the improvement process learns by counting state-action pairs. The outer loop tests the candidate scheduler against the hypothesis using SMC and is iterated until an example is found or sufficient attempts have been made. The inner loop does not in general converge to the true maximum (the number of state-actions does not actually indicate scheduler probability), but is sometimes successful because the outer loop randomly explores local maxima. This makes the number of samples used by the inner loop critical: too many may reduce the randomness of the outer loop's exploration and thus significantly reduce the probability of finding examples. A further problem is that the repeated hypothesis tests of the outer loop will eventually produce erroneous results.

In [6] the authors present learning algorithms to bound the maximum probability of reachability properties of MDPs. The algorithms work by refining upper and lower bounds associated to individual state-actions, which are initially all set to the most conservative values. Like the approaches of [15,26], the algorithms are limited to memoryless schedulers of tractable size. Unlike the approach of [15], however, the algorithms do not learn by counting the occurrence of state-actions. During simulation, when a state that satisfies the property its reached, the bounds of all the state-actions along the path that reached it are updated according to the true (or estimated) probabilities along the path. This ensures that the bounds remain correct with respect to the true optima, although convergence is very slow. Actions are initially chosen uniformly at random (as in [15]), such that the initial successful simulations will favour the "most popular" state-actions, rather than those that maximise the probability. Since the algorithms resolve nondeterminism by choosing uniformly at random an action that maximises the probability according to the current state-action bounds, the initial simulations may prevent the algorithms from providing tight bounds.

The present work builds on the elements of lightweight verification for MDPs introduced in [27]. In [27] the authors use an incremental hash function and a pseudo-random number generator to define history-dependent schedulers using only $\mathcal{O}(1)$ memory. This allows the schedulers to be selected at random and tested individually, thus facilitating Monte Carlo algorithms that are indifferent to the size of the sample space. The full details of these techniques are described in Section 4.

3 Preliminaries

In this work we make use of the following definitions.

An MDP comprises a possibly infinite set of states S, a finite set of actions A, a finite set of probabilities Q and a relation $T : S \times A \times S \times Q$, such that $\forall s \in S$ and $\forall a \in A$, $\sum_{\forall s' \in S} T(s, a, s') = r$, where $r \in \{0, 1\}$. The execution of an MDP produces a sequence of transitions between states that induces a set of traces $\Omega = S^+$. Given an MDP in state s, an action a is chosen nondeterministically from the set $\{a' \in A : \sum_{\forall s' \in S} T(s, a', s') = 1\}$. A new state $d \in S$ is then chosen at random with probability $T(s, a, d)$.

To make nondeterministic choices we assume the existence of a scheduler. A (deterministic) history-dependent scheduler is a function $\mathfrak{S} : \Omega \to A$. A (deterministic) memoryless scheduler is a function $\mathfrak{M} : S \to A$. Intuitively, at each state in the course of an execution, a history-dependent scheduler chooses an action based on the sequence of previous states and a memoryless scheduler chooses an action based only on the current state. We later mention in passing the notion of a probabilistic scheduler, which is defined by a function $\mathfrak{P} : S \times A \to \mathbb{Q}$, such that $\forall s \in S$, $\sum_{\forall a \in A} \mathfrak{P}(s, a) = r$, with $r \in \{0, 1\}$. Intuitively, in any state of an execution an action is chosen probabilistically. In what follows we use the general term 'scheduler' to mean history-dependent schedulers (which include memoryless schedulers) unless specifically qualified by the terms 'memoryless' or 'probabilistic'.

The application of a scheduler to an MDP resolves the nondeterminism and thus induces a discrete time Markov chain over which the probabilistic measure of temporal logic properties may be defined. In this work we describe sampling algorithms to find deterministic schedulers that approximately maximise or minimise the probability of such properties.

In the context of SMC, we consider finite traces generated by simulation, which are verified on the fly by an automaton that encodes the property. The mechanisms, merits and limitations of checking temporal properties on finite traces are discussed at length in the literature, e.g., in [28,11,10,12,9]. In the sequel we simply assume that there exists a function to decide whether a trace satisfies a property, that traces are of bounded length for a given property and that $\mathfrak{S}$ and $\mathfrak{M}$ are therefore of finite domain. For concreteness we define bounded linear time logical properties using the following syntax:

$$\phi = \phi \vee \phi \mid \phi \wedge \phi \mid \neg\phi \mid \mathbf{X}\phi \mid \mathbf{F}^k\phi \mid \mathbf{G}^k\phi \mid \phi \mathbf{U}^k \phi \mid \alpha. \quad (1)$$

The symbol α denotes an atomic property that is either *true* or *false* in a state. Given a trace $\omega \in \Omega$, comprising states $s_0 s_1 \ldots$, $\omega^{(i)}$ denotes the trace suffix $s_i s_{i+1} \ldots$. The satisfaction relation $\models$ over (1) is constructed inductively as follows:

$$\begin{aligned}
&\omega^{(i)} \models \mathit{true} \\
&\omega^{(i)} \models \alpha \iff \alpha \text{ evaluates } \mathit{true} \text{ in state } \omega_i \\
&\omega^{(i)} \models \neg\varphi \iff \omega^{(i)} \models \varphi \notin \models \\
&\omega^{(i)} \models \varphi_1 \vee \varphi_2 \iff \omega^{(i)} \models \varphi_1 \text{ or } \omega^{(i)} \models \varphi_2 \\
&\omega^{(i)} \models \mathbf{X}^k\varphi \iff \omega^{(k+i)} \models \varphi \\
&\omega^{(i)} \models \varphi_1 \mathbf{U}^k \varphi_2 \iff \\
&\qquad \exists j \in \{i, \ldots, i+k\} : \omega^{(j)} \models \varphi_2 \\
&\qquad \wedge\, (j = i \vee \forall l \in \{i, \ldots, j-1\} : \omega^{(l)} \models \varphi_1)
\end{aligned} \quad (2)$$

Other elements of the relation are constructed using the equivalences $\mathit{false} \equiv \neg\mathit{true}$, $\phi \wedge \phi \equiv \neg(\neg\phi \vee \neg\phi)$, $\mathbf{F}^k\phi \equiv \mathit{true}\mathbf{U}^k\phi$, $\mathbf{G}^k\phi \equiv \neg(\mathit{true}\mathbf{U}^k\neg\phi)$.

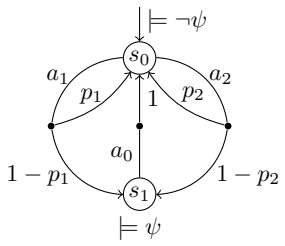


Fig. 3: MDP with different optima for general and memoryless schedulers.

Figure 3 illustrates a simple MDP for which memoryless and history-dependent schedulers give different optima for the bounded temporal logic property $\mathbf{X}(\psi \wedge \mathbf{X}\mathbf{G}^t\neg\psi)$ when $p_1 \neq p_2$ and $t > 0$. Intuitively, the property states that on the next step ψ will be *true* and, on the step after that, $\neg\psi$ will be remain *true* for t further time steps. The property is satisfied by the sequence of states $s_0 s_1 s_0 s_0 \cdots$. If $p_1 > p_2$, the maximum probability for $s_0 s_1$ is achieved with action a_2, while the maximum probability for $s_0 s_0$ is achieved with action a_1. Given that both transitions start in the same state, a memoryless scheduler will not achieve the maximum probability achievable with a history-dependent scheduler.

Statistical Model Checking with Plasma

The algorithms we present here are implemented in our SMC platform Plasma (Platform for Learning and Advanced Statistical Model checking Algorithms [5]). Plasma is modular, allowing new modelling languages, logics and algorithms to be plugged-in and take advantage of its graphical user interface, its integrated development environment and its ability to correctly divide simulations on parallel computational architectures. We introduce

here the basic notions of SMC with PLASMA applied to Markov chains.

PLASMA implicitly implements an indicator function $\mathbf{1}(\omega \models \varphi) \in \{0, 1\}$ that returns 1 iff the trace $\omega \in \Omega$ satisfies property φ, where φ is specified according to (1) and (2). This function is used to estimate with probabilistic confidence the probability of the property or to decide an hypothesis about the probability.

Typically, the probability of property φ is estimated by the proportion of traces that individually satisfy it, i.e., $\hat{p} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(\omega_i \models \varphi)$, where $\hat{p}$ denotes the estimated probability of true probability p and $\omega_1, \ldots, \omega_N$ are N independently generated simulation traces. PLASMA calculates a priori the required number of simulations according to a Chernoff bound [30], which allows the user to specify an error ε and a probability δ that the estimate $\hat{p}$ will not lie outside the interval $p \pm \varepsilon$. The Chernoff bound thus ensures $\mathrm{P}(\mid \hat{p} - p \mid \geq \varepsilon) \leq \delta$. Parameter δ is related to the number of simulations N by $\delta = 2e^{-2N\varepsilon^2}$ [30], giving

$$N = \left\lceil (\ln 2 - \ln \delta)/(2\varepsilon^2) \right\rceil . \tag{3}$$

In the case of hypothesis testing, PLASMA adopts the sequential probability ratio test (SPRT) of Wald [32] to test hypotheses of the form $\mathrm{P}(\omega \models \varphi) \bowtie \theta$, where $\bowtie \in \{\leq, \geq\}$ and θ is a user-specified probability threshold. The number of simulations required to decide the test is typically fewer than (3) but is dependent on how close θ is to the true probability. The number is therefore not known in advance. To evaluate $\mathrm{P}(\omega \models \varphi) \bowtie \theta$, the SPRT constructs hypotheses $H_0 : \mathrm{P}(\omega \models \varphi) \geq p_0$ and $H_1 : \mathrm{P}(\omega \models \varphi) \leq p_1$, where $p_0 = \theta + \varepsilon$ and $p_1 = \theta - \varepsilon$ for some user-defined interval specified by ε [32]. The SPRT also requires parameters α and β to specify, respectively, the maximum acceptable probabilities of incorrectly rejecting a true H_0 and incorrectly accepting a false H_0. To choose between H_0 and H_1, the SPRT defines the probability ratio

$$\mathit{ratio} = \prod_{i=1}^{n} \frac{(p^1)^{\mathbf{1}(\omega_i \models \varphi)}(1-p^1)^{\mathbf{1}(\omega_i \models \neg\varphi)}}{(p^0)^{\mathbf{1}(\omega_i \models \varphi)}(1-p^0)^{\mathbf{1}(\omega_i \models \neg\varphi)}},$$

where n is the number of simulation traces ω_i, $i \in \{1, \ldots, n\}$, generated so far. The test proceeds by performing a simulation and calculating *ratio* until one of two conditions is satisfied: H_1 is accepted if $\mathit{ratio} \geq (1-\beta)/\alpha$ and H_0 is accepted if $\mathit{ratio} \leq \beta/(1-\alpha)$.

Parallelisation of SMC is conceptually simple with lightweight algorithms, but balancing the simulation load on unreliable or heterogeneous computing devices must be achieved without introducing a "selection bias". The problem arises because simulation traces that satisfy a property will, in general, take a different time to generate than those which do not. If the SMC task is divided among a number of clients of different speed or reliability, a naive balancing approach will be biased in favour of results that are generated quickly. To overcome this phenomenon, PLASMA adopts the load balancing algorithm proposed in [36]. PLASMA's GUI facilitates easy parallelisation on ad hoc networked computers or on dedicated grids and clusters. The server application (an instance of PLASMA) starts the job and waits to be contacted by available clients (instances of PLASMA Service). Our estimation experiments in Section 6 were distributed on the IGRIDA computing grid[2].

4 Lightweight Verification of MDPs

In this section we recall the elemental sampling techniques of [27].

Storing schedulers as explicit mappings does not scale, so we represent schedulers using uniform pseudo-random number generators (PRNG) that are initialised with a seed value and iterated to generate the next pseudo-random number. In general, such PRNGs aim to ensure that arbitrary subsets of sequences of iterates are uniformly distributed and that consecutive iterates are statistically independent. PRNGs are commonly used to implement the uniform probabilistic scheduler, which chooses actions uniformly at random and thus explores all possible combinations of nondeterministic choices. Executing such an implementation twice with the same seed will produce identical traces. Executing the implementation with a different seed will produce an unrelated set of choices: individual deterministic schedulers cannot be identified, so it is not possible to estimate the probability of a property under a specific memoryless or history-dependent scheduler. We use a PRNG to resolve nondeterministic choices, but not to make those choices probabilistically: we use it to range over all the possible choices, such that repeated scheduler samplings will eventually consider all possible sequences of actions. We also rely on the fact that the seed of a PRNG uniquely defines the sequence of pseudo-random values. Hence, in contrast to the uniform probabilistic scheduler, actions are consistent between simulations.

An apparently plausible solution is to use independent PRNGs to resolve nondeterministic and probabilistic choices. It is then possible to generate multiple probabilistic simulation traces per scheduler by keeping the seed of the PRNG for nondeterministic choices fixed while choosing random seeds for a separate PRNG for probabilistic choices. Unfortunately, the schedulers generated by this approach do not span the full range of general or even memoryless schedulers. Since the sequence of iterates from the PRNG used for nondeterministic choices will be the same for all instantiations of the PRNG used for probabilistic choices, the i^{th} iterate of the PRNG for nondeterministic choices will always be the same, regardless of the state arrived at by the

[2] igrida.gforge.inria.fr

previous probabilistic choices. The i^{th} chosen action can be neither state nor trace dependent, as required by our definitions of memoryless and history-dependent schedulers, respectively.

4.1 General Schedulers Using Hash Functions

We therefore construct a per-step PRNG seed that is a *hash* of the integer identifying a specific scheduler concatenated with an integer representing the sequence of states up to the present.

We assume that a state of an MDP is an assignment of values to a vector of system variables $v_i, i \in \{1, \ldots, n\}$. Each v_i is represented by a number of bits b_i, typically corresponding to a primitive data type (*int*, *float*, *double*, etc.). The state can thus be represented by the concatenation of the bits of the system variables, such that a sequence of states may be represented by the concatenation of the bits of all the states. Without loss of generality, we interpret such a sequence of states as an integer of $\sum_{i=1}^{n} b_i$ bits, denoted $\overline{s}$, and refer to this in general as the *trace vector*. A scheduler is denoted by an integer σ, which is concatenated to $\overline{s}$ (denoted $\sigma : \overline{s}$) to uniquely identify a trace and a scheduler. Our approach is to generate a hash code $h = \mathcal{H}(\sigma : \overline{s})$ and to use h as the seed of a PRNG that resolves the next nondeterministic choice.

The hash function $\mathcal{H}$ thus maps $\sigma : \overline{s}$ to a seed that is deterministically dependent on the trace and the scheduler. The PRNG maps the seed to a value that is uniformly distributed but nevertheless deterministically dependent on the trace and the scheduler. In this way we approximate the deterministic functions $\mathfrak{S}$ and $\mathfrak{M}$ described in Section 3. The (potential) approximation arises because there may be more possible schedulers than can be uniquely identified by the bits of σ. Importantly, the standard properties of hash functions and PRNGs serve to ensure that there is no systematic bias. The hash function is expected to map a large set of integers to a smaller set of integers such that sequential or otherwise related input values have low probability of collision. Sequential iterates of the PRNG are expected to be (pseudo) statistically independent and (pseudo) uniformly distributed. Hence, if σ is chosen uniformly at random, the probability of taking a particular action in a state (or following a sequence of states) will be (pseudo) uniformly distributed among the enabled actions.

Algorithm 1 is the basic simulation function used by our algorithms.

Algorithm 1: Simulate

Input:
$\mathcal{M}$: an MDP with initial state s_0
φ: a property
σ: an integer identifying a scheduler
Output:
ω: a simulation trace

```
1  Let U_prob, U_nondet be uniform PRNGs with respective samples r_pr, r_nd
2  Let H be a hash function
3  Let s denote a state, initialised s ← s_0
4  Let ω denote a trace, initialised ω ← s
5  Let s̄ be the trace vector, initially empty
6  Set seed of U_prob randomly
7  while ω ⊨ φ is not decided do
8      s̄ ← s̄ : s
9      Set seed of U_nondet to H(σ : s̄)
10     Iterate U_nondet to generate r_nd and use to resolve nondeterministic choice
11     Iterate U_prob to generate r_pr and use to resolve probabilistic choice
12     Set s to the next state
13     ω ← ω : s
```

4.2 An Efficient Iterative Hash Function

To implement our approach, we use an efficient hash function that constructs seeds incrementally. The function is based on modular division [20, Ch. 6], such that $h = (\sigma : \overline{s}) \bmod m$, where m is a suitably large prime.

Since $\overline{s}$ is a concatenation of states, it is usually very much larger than the maximum size of integers supported as primitive data types. Hence, to generate h we use Horner's method [17][20, Ch. 4]: we set $h_0 = \sigma$ and find $h \equiv h_n$ (n as in Section 4.1) by iterating the recurrence relation

$$h_i = (h_{i-1} 2^{b_i} + v_i) \bmod m. \tag{4}$$

The size of m defines the maximum number of different hash codes. The precise value of m controls how the hash codes are distributed. To avoid collisions, a simple heuristic is that m should be a large prime not close to a power of 2 [8, Ch. 11]. The number of schedulers is typically much larger than the number of possible hash codes, hence collisions are theoretically inevitable. This means that not all possible schedulers are realisable with a given hash function and PRNG. We suppose, however, that there is no scheduler that cannot be realised with *some* hash function and PRNG. The problem of collisions can thus be conceivably addressed by also choosing the hash function and PRNG at random. A scheduler would then be defined by its label, its hash function and its PRNG. We do not implement this idea here to avoid unnecessary complication and because collisions are not the principal limitation. There are typically many orders of magnitude more seeds than we can test, hence the problem of finding the best available scheduler supersedes the problem that the best available scheduler may not be optimal. We anticipate that our proposed solutions to accelerate convergence (property-focused scheduler space and piecewise construction of schedulers) will effectively bypass the collision problem.

In practical implementations it is an advantage to perform calculations using primitive data types that are native to the computational platform, so the sum in (4) should always be less than or equal to the maximum permissible value. To achieve this, given $x, y, m \in \mathbb{N}$, we note the following congruences:

$$(x + y) \bmod m \equiv (x \bmod m + y \bmod m) \bmod m \quad (5)$$
$$(xy) \bmod m \equiv ((x \bmod m)(y \bmod m)) \bmod m \quad (6)$$

The addition in (4) can thus be re-written in the form of (5), such that each term has a maximum value of $m-1$:

$$h_i = ((h_{i-1}2^{b_i}) \bmod m + (v_i) \bmod m) \bmod m \quad (7)$$

To prevent overflow, m must be no greater than half the maximum possible integer. Re-writing the first term of (7) in the form of (6), we see that before taking the modulus it will have a maximum value of $(m-1)^2$, which will exceed the maximum possible integer. To avoid this, we take advantage of the fact that h_{i-1} is multiplied by a power of 2 and that m has been chosen to prevent overflow with addition. We thus apply the following recurrence relation:

$$\begin{aligned}(h_{i-1}2^{j}) \bmod m = {} & (h_{i-1}2^{j-1}) \bmod m \\ & + (h_{i-1}2^{j-1}) \bmod m \end{aligned} \quad (8)$$

Equation (8) allows our hash function to be implemented using efficient native arithmetic. Moreover, we infer from (4) that to find the hash code corresponding to the current state in a trace, we need only know the current state and the hash code from the previous step. When considering memoryless schedulers we need only know the current state.

4.3 Hypothesis Testing Multiple Schedulers

We apply the SPRT to multiple (randomly chosen) schedulers to test hypotheses of the form *there exists a scheduler such that* $\mathrm{P}(\omega \models \varphi) \bowtie p$. To test hypotheses of the form *there is no scheduler such that* $\mathrm{P}(\omega \models \varphi) \bowtie p$, our algorithm simply searches for a scheduler that disproves the hypothesis. Since the probability of error with the SPRT applied to multiple hypotheses is cumulative, we consider the probability of no errors in any of M tests. Hence, in order to ensure overall error probabilities α and β, we adopt $\alpha_M = 1 - \sqrt[M]{1-\alpha}$ and $\beta_M = 1 - \sqrt[M]{1-\beta}$ in our stopping conditions. H_1 is accepted if $ratio \geq (1-\beta_M)/\alpha_M$ and H_0 is accepted if $ratio \leq \beta_M/(1-\alpha_M)$. Algorithm 2 demonstrates the sequential hypothesis test for multiple schedulers. If the algorithm finds an example, the hypothesis is true with at least the specified confidence.

Algorithm 2: SPRT for multiple schedulers

Input:
$\mathcal{M}, \varphi$: the MDP and property of interest
$H \in \{H_0, H_1\}$: the hypothesis with interval $\theta \pm \varepsilon$
α, β: the desired error probabilities of H
M: the maximum number of schedulers to test
Output: A scheduler that satisfies H or an inconclusive result

Let $p^0 = \theta + \varepsilon$ and $p^1 = \theta - \varepsilon$ be the bounds of H
Let $\alpha_M = 1 - \sqrt[M]{1-\alpha}$ and $\beta_M = 1 - \sqrt[M]{1-\beta}$
Let $A = (1-\beta_M)/\alpha_M$ and $B = \beta_M/(1-\alpha_M)$
Let $\mathcal{U}_{\text{seed}}$ be a uniform PRNG and σ be its sample
for $i \in \{1, \ldots, M\}$ *while* H *is not accepted* **do**
  Iterate $\mathcal{U}_{\text{seed}}$ to generate σ_i
  Let $ratio = 1$
  while $ratio > A \wedge ratio < B$ **do**
    $\omega \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma_i)$
    $ratio \leftarrow \frac{(p^1)^{\mathbf{1}(\omega \models \varphi)}(1-p^1)^{\mathbf{1}(\omega \models \neg\varphi)}}{(p^0)^{\mathbf{1}(\omega \models \varphi)}(1-p^0)^{\mathbf{1}(\omega \models \neg\varphi)}} ratio$
  if $ratio \leq A \wedge H = H_0 \vee ratio \geq B \wedge H = H_1$ **then**
    accept H

4.4 Estimating Multiple Schedulers

We consider the strategy of sampling M schedulers to estimate the maximum or minimum probabilities of satisfying a property. We thus generate M estimates $\{\hat{p}_1, \ldots, \hat{p}_M\}$ corresponding to true values $\{p_1, \ldots, p_M\}$, and take either the maximum ($\hat{p}_{\max}$) or minimum ($\hat{p}_{\min}$), as required. To overcome the cumulative probability of error with the standard Chernoff bound, we specify that *all* estimates $\hat{p}_i$ must be within ε of their respective true values p_i, ensuring that any $\hat{p}_{\min}, \hat{p}_{\max} \in \{\hat{p}_1, \ldots, \hat{p}_M\}$ are within ε of their true value. Given that all estimates $\hat{p}_i$ are statistically independent, the probability that they are all less than their upper bound is expressed by

$$\mathrm{P}(\bigwedge_{i=1}^{M} \hat{p}_i - p_i \leq \varepsilon) \geq (1 - e^{-2N\varepsilon^2})^M.$$

Hence, $\mathrm{P}(\bigvee_{i=1}^{M} \hat{p}_i - p_i \geq \varepsilon) \leq 1 - (1 - e^{-2N\varepsilon^2})^M$, giving

$$N = \left\lceil -\ln\left(1 - \sqrt[M]{1-\delta}\right)/(2\varepsilon^2) \right\rceil.$$

This ensures that $\mathrm{P}(p_{\min} - \hat{p}_{\min} \geq \varepsilon) \leq \delta$ and $\mathrm{P}(\hat{p}_{\max} - p_{\max} \geq \varepsilon) \leq \delta$. To ensure the usual stronger conditions that $\mathrm{P}(\mid p_{\max} - \hat{p}_{\max} \mid \geq \varepsilon) \leq \delta$ and $\mathrm{P}(\mid p_{\min} - \hat{p}_{\min} \mid \geq \varepsilon) \leq \delta$, we have

$$N = \left\lceil \left(\ln 2 - \ln\left(1 - \sqrt[M]{1-\delta}\right)\right)/(2\varepsilon^2) \right\rceil. \quad (9)$$

N scales logarithmically with M, making it tractable to consider many schedulers. In the case of $M = 1$, (9) degenerates to (3). Note, however, that the confidence

expressed by (9) is with respect to the sampled set, not with respect to the true extrema.

Algorithm 3 is the resulting extremal probability estimation algorithm for multiple schedulers. Note that the algorithm distinguishes $p_{\min}, p_{\max}$ (the notional true extreme probabilities), $p_{\underline{\min}}, p_{\overline{\max}}$ (the true probabilities for the schedulers chosen by the algorithm) and $\hat{p}_{\underline{\min}}, \hat{p}_{\overline{\max}}$ (the estimated probabilities using the chosen schedulers).

Algorithm 3: Estimation with multiple schedulers

Input:
$\mathcal{M}, \varphi$: the MDP and property of interest
ε, δ: the required Chernoff bound
M: the number of schedulers to test

Output: $\hat{p}_{\underline{\min}} \approx p_{\min}$, $\hat{p}_{\overline{\max}} \approx p_{\max}$, where
$p_{\underline{\min}} \geq p_{\min}$, $p_{\overline{\max}} \leq p_{\max}$ and
$\mathrm{P}(|p_{\underline{\min}} - \hat{p}_{\underline{\min}}| \geq \epsilon) \leq \delta$,
$\mathrm{P}(|p_{\overline{\max}} - \hat{p}_{\overline{\max}}| \geq \epsilon) \leq \delta$

```
Let N = ⌈ln(2/(1 − ᴹ√(1 − δ)))/(2ε²)⌉ be the no. of simulations per scheduler
Let U_seed be a uniform PRNG and σ its sample
Initialise p̂_min ← 1 and p̂_max ← 0
Set seed of U_seed randomly
for i ∈ {1, ..., M} do
    Iterate U_seed to generate σ_i
    Let truecount = 0 be the initial number of traces that satisfy φ
    for j ∈ {1, ..., N} do
        ω_j ← Simulate(M, φ, σ_i)
        truecount ← truecount + 1(ω_j ⊨ φ)
    Let p̂_i = truecount/N
    if p̂_max < p̂_i then
        p̂_max = p̂_i
    if p̂_i > 0 ∧ p̂_min > p̂_i then
        p̂_min = p̂_i
if p̂_max = 0 then
    No schedulers were found to satisfy φ
```

Figure 4 shows the empirical cumulative distribution of schedulers generated by Algorithm 3 applied to the MDP of Fig. 3, using $p_1 = 0.9$, $p_2 = 0.5$, $\varphi = \mathbf{X}(\psi \wedge \mathbf{XG}^4 \neg\psi)$, $\varepsilon = 0.01$, $\delta = 0.01$ and $M = 300$. The vertical lines mark the true probabilities of φ under each of the history-dependent and memoryless schedulers (indicated by arrows). The shaded areas show the $\pm\varepsilon$ error bounds, relative to the true probabilities. There are multiple estimates per scheduler, but all estimates are within their respective confidence bounds.

5 Smart Sampling

The simple sampling strategies used by Algorithms 2 and 3 have the disadvantage that they allocate equal simulation budget to all schedulers, regardless of their merit. In general, the problem we address has two independent components: the rarity of near optimal schedulers and the probability of the property under a near optimal scheduler. We should allocate our simulation budget accordingly and not waste budget on schedulers that are clearly not optimal.

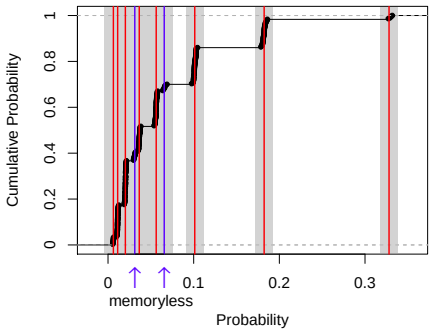


Fig. 4: Empirical cumulative distribution of estimates from Algorithm 3 applied to MDP of Fig. 3.

Motivated by the above, our smart estimation algorithm comprises three stages: (*i*) an initial undirected sampling experiment to discover the nature of the problem, (*ii*) a targeted sampling experiment to generate a candidate set of schedulers with high probability of containing an optimal scheduler and (*iii*) iterative refinement of the candidates to estimate the probability of the best scheduler with specified confidence. By excluding the schedulers with the worst estimated probabilities and re-allocating their simulation budget to the schedulers that remain, at each iterative step of stage (*iii*) the number of schedulers reduces while the confidence of their estimates increases. With a suitable choice of per-iteration budget, the algorithm is guaranteed to terminate.

In the following subsection we develop the theoretical basis of stage (*ii*).

5.1 Maximising the Probability of Seeing a Good Scheduler

We assume the existence of an MDP and a bounded property φ whose probability we wish to maximise by choosing a suitable scheduler from the finite set $\mathfrak{S}$. Let $\mathcal{P} : \mathfrak{S} \to [0, 1]$ be a function mapping schedulers to their probability of satisfying φ and let $p_{\max} = \max_{\sigma \in \mathfrak{S}}(\mathcal{P}(\sigma))$. For the sake of exposition, we consider the problem of finding a scheduler that maximises the probability of satisfying φ and define a "good" (near optimal) scheduler to be one in the set $\mathfrak{S}_g = \{\sigma \in \mathfrak{S} \mid \mathcal{P}(\sigma) \geq p_{\max} - \varepsilon\}$ for some $\varepsilon \in (0, p_{\max}]$. Intuitively, a good scheduler is one whose probability of satisfying φ is within ε of $p_{\max}$, noting that we may similarly define a good scheduler to be one within ε of $p_{\min} = \min_{\sigma \in \mathfrak{S}}(\mathcal{P}(\sigma))$, or to be in any

other subset of $\mathfrak{S}$. In particular, to address reward-based MDP optimisations, a good scheduler could be defined to be the subset of $\mathfrak{S}$ that is near optimal with respect to a reward scheme. The notion of a "best" scheduler follows intuitively from the definition of a good scheduler.

Given that we sample uniformly from $\mathfrak{S}$, the probability of finding a good scheduler is $p_g = |\mathfrak{S}_g|/|\mathfrak{S}|$. The average probability of a good scheduler is $p_{\overline{g}} = \sum_{\sigma\in\mathfrak{S}_g} \mathcal{P}(\sigma)/|\mathfrak{S}_g|$. If we select M schedulers at random and verify each with N simulations, the expected number of traces that satisfy φ using a good scheduler is thus $Mp_gNp_{\overline{g}}$. The probability of seeing a trace that satisfies φ using a good scheduler is the cumulative probability

$$(1-(1-p_g)^M)(1-(1-p_{\overline{g}})^N). \tag{10}$$

Hence, for a given simulation budget $N_{\max} = NM$, to implement stage (ii) the idea is to choose N and M to maximise (10) and keep any scheduler that produces at least one trace that satisfies φ. Since, a priori, we are generally unaware of even the magnitudes of p_g and $p_{\overline{g}}$, stage (i) is necessarily uninformed and we set $N = M = \lceil\sqrt{N_{\max}}\rceil$. The results of stage (i) allow us to estimate p_g and $p_{\overline{g}}$ (see Fig. 9a) and thus maximise (10). This may be done numerically, but we have found the heuristic $N = \lceil 1/p_{\overline{g}}\rceil$ to be near optimal in all but extreme cases.

5.2 Smart Estimation

Algorithm 4 is our smart estimation algorithm to find schedulers that maximise the probability of a property. The algorithm to find minimising schedulers is similar. As with Algorithm 3, Algorithm 4 distinguishes $p_{\max}$ (the notional true maximum probability), $p_{\overline{\max}}$ (the true probability using the current best candidate scheduler) and $\hat{p}_{\overline{\max}}$ (the estimated probability using the best candidate scheduler).

Lines 1 to 5 implement stage (i): N and M are set equal, simulation experiments are performed and the maximum estimate $\hat{p}_{\overline{\max}}$ is found. Lines 6 to 10 implement stage (ii): the initial candidate set of schedulers is generated by setting $N = \lceil 1/\hat{p}_{\overline{\max}}\rceil$ and removing schedulers that produce no traces that satisfy the property. Lines 11 to 23 implement stage (iii). The inner loop (lines 16 to 19) requests simulations and exits as soon as the number of simulations is sufficient for the required confidence or when the maximum number for the iteration has been reached. Lines 20 to 23 calculate the estimates and select the upper quantile of schedulers for the next iteration. The outer loop (line 12) quits once the set of estimates are known with the required confidence.

The per-iteration simulation budget $N_{\max}$ must be greater than the number needed by the standard Chernoff bound (3), to ensure that there will be sufficient simulations to guarantee the specified confidence if the algorithm refines the candidate set to a single scheduler. Typically, the per-iteration budget will be greater than this, such that the required confidence is reached before refining the set of schedulers to a single element. Under these circumstances the confidence is judged according to the Chernoff bound for multiple estimates (9).

Algorithm 4 may be further optimised by re-using the simulation results from previous iterations of stage (iii). The contribution is small, however, because confidence decreases exponentially with the age (in terms of iterations) of the results.

Algorithm 4: Smart Estimating

Input:
$\mathcal{M}$: an MDP
φ: a property
ϵ, δ: the required Chernoff bound
$N_{\max} > \ln(2/\delta)/(2\epsilon^2)$: the per-iteration budget
Output: $\hat{p}_{\overline{\max}} \approx p_{\max}$, where $p_{\overline{\max}} \le p_{\max}$ and $\mathrm{P}(|p_{\overline{\max}} - \hat{p}_{\overline{\max}}| \ge \epsilon) \le \delta$

1 $N \leftarrow \lceil\sqrt{N_{\max}}\rceil$; $M \leftarrow \lceil\sqrt{N_{\max}}\rceil$
2 $S \leftarrow \{M$ seeds chosen uniformly at random$\}$
3 $\forall\sigma \in S, \forall i \in \{1,\dots,N\} : \omega_i^\sigma \leftarrow \mathrm{Simulate}(\mathcal{M},\varphi,\sigma)$
4 $R : S \to \mathbb{N}$ maps scheduler seeds to number of traces satisfying φ:
$R \leftarrow \{(\sigma,n) \mid \sigma \in S \wedge \mathbb{N} \ni n = \sum_{i=1}^{N} \mathbf{1}(\omega_i^\sigma \models \varphi)\}$
5 $\hat{p}_{\overline{\max}} \leftarrow \max_{\sigma\in S}(R(\sigma)/N)$
6 $N \leftarrow \lceil 1/\hat{p}_{\overline{\max}}\rceil$, $M \leftarrow \lceil N_{\max}\,\hat{p}_{\overline{\max}}\rceil$
7 $S \leftarrow \{M$ seeds chosen uniformly at random$\}$
8 $\forall\sigma \in S, \forall i \in \{1,\dots,N\} : \omega_i^\sigma \leftarrow \mathrm{Simulate}(\mathcal{M},\varphi,\sigma)$
9 $R \leftarrow \{(\sigma,n) \mid \sigma \in S \wedge \mathbb{N} \ni n = \sum_{i=1}^{N} \mathbf{1}(\omega_i^\sigma \models \varphi)\}$
10 $S \leftarrow \{\sigma \in S \mid R(\sigma) > 0\}$
11 $\forall\sigma \in S,\ R(\sigma) \leftarrow 0$; $i \leftarrow 0$; $conf \leftarrow 1$
12 **while** $conf > \delta \wedge S \neq \emptyset$ **do**
13 $\quad i \leftarrow i + 1$
14 $\quad M_i \leftarrow |S|$
15 $\quad N_i \leftarrow 0$
16 $\quad$ **while** $conf > \delta \wedge N_i < \lceil N_{\max}/M_i\rceil$ **do**
17 $\quad\quad N_i \leftarrow N_i + 1$
18 $\quad\quad conf \leftarrow 1 - (1 - e^{-2\epsilon^2 N_i})^{M_i}$
19 $\quad\quad \forall\sigma \in S : \omega_{N_i}^\sigma \leftarrow \mathrm{Simulate}(\mathcal{M},\varphi,\sigma)$
20 $\quad R \leftarrow \{(\sigma,n) \mid \sigma \in S \wedge \mathbb{N} \ni n = \sum_{j=1}^{N_i} \mathbf{1}(\omega_j^\sigma \models \varphi)\}$
21 $\quad \hat{p}_{\overline{\max}} \leftarrow \max_{\sigma\in S}(R(\sigma)/N_i)$
22 $\quad R' : \{1,\dots,|S|\} \to S$ is an injective function s.t.
$\forall(n,\sigma),(n',\sigma') \in R',\ n > n' \implies R(\sigma) \ge R(\sigma')$
23 $\quad S \leftarrow \{\sigma \in S \mid \sigma = R'(n) \wedge n \in \{\lfloor|S|/2\rfloor,\dots,|S|\}\}$

5.3 Smart Hypothesis Testing

We wish to test the hypothesis that there exists a scheduler such that property φ has probability $\bowtie \theta$, where $\bowtie \in \{\ge,\le\}$. Two advantages of sequential hypothesis testing are that it is not necessary to estimate the actual probability to know if an hypothesis is satisfied, and the easier the hypothesis is to satisfy, the quicker it is to get a result. Algorithm 5 maintains these advantages and

uses smart sampling to improve on the performance of Algorithm 2. For the purposes of exposition, Algorithm 5 tests H_0, as described in Section 3. The algorithm to test H_1 is similar.

A sub-optimal approach would be to simply use Algorithm 4 to refine a set of schedulers until one is found whose estimate satisfies the hypothesis with confidence according to a Chernoff bound. We improve on this with sequential hypothesis testing, using the results given in Section 4.3 and as applied in Algorithm 2. Algorithm 5 refines a set of schedulers according to their estimated probability in the same manner as Algorithm 4, but also uses the simulation results to test each scheduler with respect to an hypothesis test for multiple schedulers. This allows the algorithm to terminate quickly when the hypothesis is easily satisfied.

We also include a further refinement. In general, smart sampling exploits the fact that the *average* estimate at each iteration is known with high confidence, i.e., confidence given by the total simulation budget. This comes from the linearity of expectation and the result of [16], where the bound is specified for a sum of arbitrary random variables, not necessarily with identical expectations. It follows that the sequential probability ratio test may also be applied to the sum of results produced during the course of an iteration. This is because the distribution of the total number of successes after a number of sequential hypothesis tests is equivalent to the distribution of successes obtained with the same total number of trials performed on the weighted average probability of the individual unknown probabilities (the weights being the number of trials on the individual tests). By the convexity of the weighted average, if the hypothesis is satisfied with respect to the total number of trials, there exists a scheduler whose probability satisfies the hypothesis with equal or better confidence.

In summary, if the "average scheduler" or an individual scheduler ever satisfies the hypothesis (lines 23 and 24), the algorithm immediately terminates and reports that the hypothesis is satisfied with the specified confidence. If all schedulers falsify the hypothesis (line 27) the algorithm terminates and reports that no scheduler in the candidate set satisfies the hypothesis. Note that this outcome does not imply that no scheduler exists that will satisfy the hypothesis, only that no scheduler was found with the given budget. If neither of the previous conditions apply, the algorithm terminates with an inconclusive result: there exists a scheduler in the candidate set that does not reject the hypothesis given the parameters.

We implement one further important optimisation. We use the threshold probability θ to directly define the simulation budget to generate the candidate set of schedulers, i.e. $N = \lceil 1/\theta \rceil$, $M = \lceil \theta N_{\max} \rceil$ (line 3). This is justified because we need only find schedulers whose probability of satisfying φ is greater than θ. By setting $N = \lceil 1/\theta \rceil$, (10) ensures that such schedulers, if they exist, have high probability of being observed. The initial uninformed exploration (stage (i)) used in Algorithm 4 is thus not necessary.

Algorithm 5 is our smart hypothesis testing algorithm. Note that we do not set a precise minimum per-iteration simulation budget because we expect the hypothesis to be decided with many fewer simulations than would be required to estimate the probability. In practice it is expedient to initially set a low per-iteration budget (e.g., 1000) and repeat the algorithm with an increased budget (e.g., increased by an order of magnitude) if the previous test was inconclusive.

6 Case Studies

To demonstrate the performance of smart sampling we have implemented Algorithms 4 and 5 in our statistical model checking platform PLASMA [5]. We performed a number of experiments on standard models taken from the numerical model checking literature, most of which can be found illustrated on the PRISM website[3]. We found that all of our estimation experiments achieved their specified Chernoff bounds ($\varepsilon = \delta = 0.01$ in all cases) with a relatively modest per-iteration simulation budget of 10^5 simulations. The actual number of simulation cores used for the estimation results was subject to availability and varied between experiments. To facilitate comparisons, in what follows we normalise all timings to be with with respect to 64 cores. Typically, each data point was produced in a few tens of seconds. Our hypothesis tests were performed on a single machine, without distribution. Despite this, most experiments completed in just a few seconds (some in fractions of a second), demonstrating that our smart hypothesis testing algorithm is able to take advantage of easy hypotheses.

6.1 IEEE 802.11 Wireless LAN Protocol

We consider a reachability property of the IEEE 802.11 Wireless LAN (WLAN) protocol, using the discrete time (MDP) model of [25]. The protocol aims to avoid "collisions" between devices sharing a communication channel, by means of an exponential backoff procedure when a collision is detected. We therefore estimate the probability of the second collision at various time steps, using Algorithm 4 with per-iteration budget of 10^5 simulations. Figure 5 illustrates the estimated maximum probabilities ($\hat{p}_{\max}$) and minimum probabilities ($\hat{p}_{\min}$) for time steps $k \in \{0, 10, \ldots, 100\}$. The property is expressed as $\mathbf{F}^k col = 2$. The shaded areas indicate the true probabilities ± 0.01, the specified absolute error bound using Chernoff bound $\varepsilon = \delta = 0.01$. Our results are clearly very close to the true values. Table 1 gives the

[3] www.prismmodelchecker.org/casestudies/

Algorithm 5: Smart Hypothesis Testing

Input:
$\mathcal{M}$: an MDP
φ: a property
$H_0 : \mathrm{P}(\omega \models \varphi) \geq \theta \pm \varepsilon$ is the hypothesis
α, β: the desired error probabilities of H_0
$N_{\max}$: the per-iteration simulation budget
Output: A scheduler that satisfies H_0 or an inconclusive result

Let $p^0 = \theta + \varepsilon$, $p^1 = \theta - \varepsilon$
Let $A = (1-\beta)/\alpha$, $B = \beta/(1-\alpha)$
$N \leftarrow \lceil 1/\theta \rceil$; $M \leftarrow \lceil \theta N_{\max} \rceil$
$S \leftarrow \{M$ seeds chosen uniformly at random$\}$
$\forall \sigma \in S, \forall i \in \{1, \ldots, N\} : \omega_i^\sigma \leftarrow \mathrm{Simulate}(\mathcal{M}, \varphi, \sigma)$
$R \leftarrow \{(\sigma, n) \mid \sigma \in S \wedge \mathbb{N} \ni n = \sum_{i=1}^{N} \mathbf{1}(\omega_i^\sigma \models \varphi)\}$
if $\frac{(p^1)^{\sum R(\sigma)}(1-p^1)^{N_{\max}-\sum R(\sigma)}}{(p^0)^{\sum R(\sigma)}(1-p^0)^{N_{\max}-\sum R(\sigma)}} \leq A$ **then**
  Accept H_0 and quit
$S \leftarrow \{\sigma \in S \mid R(\sigma) > 0\}$, $M \leftarrow |S| + 1$
while $M > 1$ **do**
  $M \leftarrow |S|$
  Let $\alpha_M = 1 - \sqrt[M]{1-\alpha}$, $\beta_M = 1 - \sqrt[M]{1-\beta}$
  Let $A_M = (1-\beta_M)/\alpha_M$, $B_M = \beta_M/(1-\alpha_M)$
  Let $ratio = 1$
  for $\sigma_i \in S, i \in \{1, \ldots, M\}$ **do**
    Let $ratio_i = 1$
    for $j \in \{1, \ldots, N\}$ **do**
      $\omega \leftarrow \mathrm{Simulate}(\mathcal{M}, \varphi, \sigma_i)$
      if $\omega \models \varphi$ **then**
        $ratio \leftarrow \frac{p_1}{p_0} ratio$; $ratio_i \leftarrow \frac{p_1}{p_0} ratio_i$
      else
        $ratio \leftarrow \frac{1-p_1}{1-p_0} ratio$; $ratio_i \leftarrow \frac{1-p_1}{1-p_0} ratio_i$
      if $ratio \leq A \vee ratio_i \leq A_M$ **then**
        Accept H_0 and quit: a scheduler exists
      if $ratio_i \geq B_M$ **then**
        Reject H_0 for σ_i and quit this loop
  if *All schedulers rejected* H_0 **then**
    Quit: no scheduler in candidates satisfies H_0
  $R' : \{1, \ldots, |S|\} \to S$ is an injective function s.t. $\forall (n, \sigma), (n', \sigma') \in R',\ n > n' \implies R(\sigma) \geq R(\sigma')$
  $S \leftarrow \{\sigma \in S \mid \sigma = R'(n) \wedge n \in \{\lfloor |S|/2 \rfloor, \ldots, |S|\}\}$
A scheduler exists that does not reject H_0 with the specified α, β and ε

results of hypothesis tests based on the same model using property $\mathbf{F}^{100} col = 2$. See Section 6.2 for a description.

The results illustrated in Fig. 5 refer to the same property and confidence as the those shown in Fig. 4 of [27]. The total simulation cost to generate a point in Fig. 5 is 1.2×10^6 (12 iterations of 10^5 simulations using smart sampling), compared to a cost of 2.7×10^8 per point in Fig. 4 of [27] (4000 schedulers tested with 67937 simulations using simple sampling). This demonstrates a more than 200-fold improvement in performance.

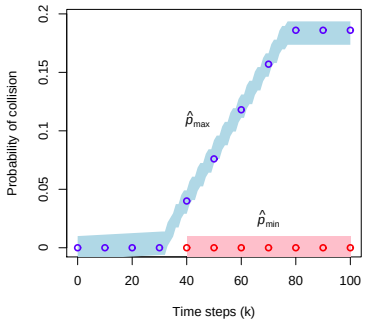


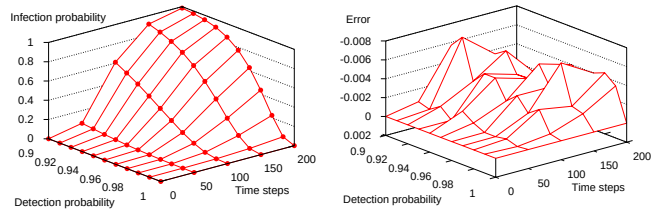
Fig. 5: Estimated maximum and minimum probabilities of second collision in WLAN protocol (circles). Shaded regions denote true values ±0.01.

CSMA 3 4	θ	0.5	0.8	0.85	0.86	0.9	0.95
	time	0.5	3.5	737	*	2.9	2.5
CSMA 3 6	θ	0.3	0.4	0.45	0.48	0.5	0.8
	time	1.3	5.2	79	*	39	2.6
CSMA 4 4	θ	0.5	0.7	0.8	0.9	0.93	0.95
	time	0.2	0.3	4.0	8.6	*	3.8
WLAN 5	θ	0.1	0.15	0.18	0.2	0.25	0.5
	time	0.8	2.6	*	2.9	2.9	1.3
WLAN 6	*time*	1.3	2.2	*	6.5	1.3	1.3

Table 1: Hypothesis test results for CSMA/CD and WLAN protocols. θ is the threshold probability or the true probability (marked by asterisk). *time* is simulation time in seconds to achieve the correct result on a single machine.

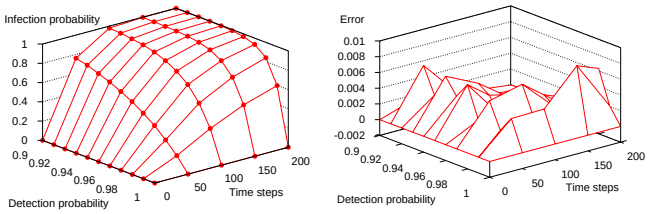
6.2 IEEE 802.3 CSMA/CD Protocol

The IEEE 802.3 CSMA/CD protocol is a wired network protocol that is similar in operation to that of IEEE 802.11, but using collision detection instead of collision avoidance. In Table 1 we give the results of applying Algorithm 5 to the IEEE 802.3 CSMA/CD protocol model of [23]. The models and parameters are chosen to compare with results given in Table III in [15], hence we also give results for hypothesis tests performed on the WLAN model used in Section 6.1. In contrast to the results of [15], our results are produced on a single machine, with no parallelisation. There are insufficient details given about the experimental conditions in [15] to make a formal comparison (e.g., error probabilities of the hypothesis tests and number of simulation cores), but it seems that the performance of our algorithm is generally much better. We set $\alpha = \beta = \delta = 0.01$, which constitute a fairly tight bound, and note that, as expected, the simulation times tend to increase as the threshold θ approaches the true probability.



(a) Estimates. (b) Errors.

Fig. 6: Minimum probability of network infection.



(a) Estimates. (b) Errors.

Fig. 7: Maximum probability of network infection.

6.3 Choice Coordination

To demonstrate the scalability of our approach we consider the choice coordination model of [29] and estimate the minimum probability that a group of six tourists will meet within T steps. The model has a parameter ($BOUND$) that limits the state space. We set $BOUND = 100$, making the state space of $\approx 5 \times 10^{16}$ states intractable to numerical model checking. Fortunately, it is possible to infer the correct probabilities from tractable parametrisations. For $T = 20$ and $T = 25$ the true minimum probabilities are respectively 0.5 and 0.75. Using smart sampling and a Chernoff bound of $\varepsilon = \delta = 0.01$, we correctly estimate the probabilities to be 0.496 and 0.745 in a few tens of seconds on 64 simulation cores.

6.4 Network Virus Infection

Network virus infection is a subject of increasing relevance. Hence, using a per-iteration budget of 10^5 simulations, we demonstrate the performance of Algorithm 4 on the PRISM virus infection case study based on [24]. The network is illustrated in Fig. 1 and comprises three sets of linked nodes: a set of nodes containing one infected by a virus, a set of nodes with no infected nodes and a set of barrier nodes which divides the first two sets. A virus chooses which node to infect nondeterministically. A node detects a virus probabilistically and we vary this probability as a parameter for barrier nodes. We consider time as a second parameter. Figures 6 and 7 illustrate the estimated probabilities that the target node in the uninfected set will be infected. We observe in Figs. 6b and 7b that the estimated minimums are within $[-0.0070, +0.00012]$ and the estimated maximums are within $[-0.00012, +0.0083]$ of their true values. The respective negative and positive biases to these error ranges reflects the fact that Algorithm 4 converges from respectively below and above (as illustrated in Fig. 9b). The average time to generate a point in Fig. 6 was approximately 100 seconds using 64 simulation cores. Points in Fig. 7 took on average approximately 70 seconds.

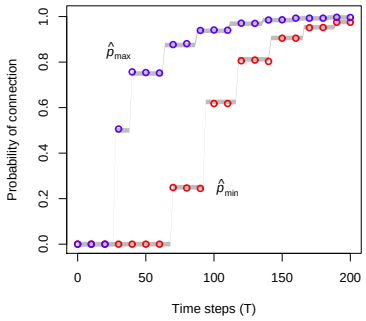


Fig. 8: Estimated probabilities that maximum path length is < 4 in gossip protocol model. Shaded regions denote ± 0.01 of true values.

6.5 Gossip Protocol

Gossip protocols are an important class of network algorithms that rely on local connectivity to propagate information globally. Using the gossip protocol model of [22], we used Algorithm 4 with per-simulation budget of 10^5 simulations to estimate the maximum ($\hat{p}_{\max}$) and minimum ($\hat{p}_{\min}$) probabilities that the maximum path length between any two nodes is less than 4 after T time steps. This is expressed by property $\mathbf{F}^T max\_path\_len < 4$. The results are illustrated in Fig. 8. Estimates of maximum probabilities are within $[-0, +0.0095]$ of the true values. Estimates of minimum probabilities are within $[-0.007, +0]$ of the true values. Each point in the figure took on average approximately 60 seconds to generate using 64 simulation cores.

7 Convergence and Counterexamples

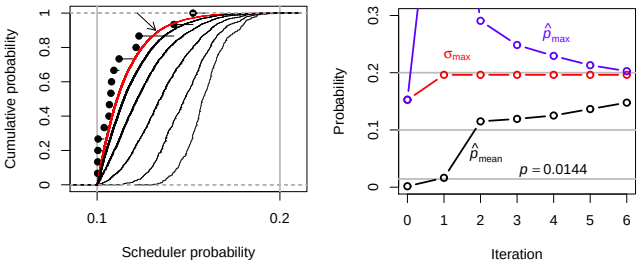
The techniques described in the preceding sections open up the possibility of efficient lightweight verification of MDPs, with the consequent possibility to take full advantage of parallel computational architectures, such as multi-core processors, clusters, grids, clouds and general purpose computing on graphics processors (GPGPU). These architectures may potentially divide the problem by the number of available computational devices (i.e., linearly), however this must be considered in the context

of scheduler space increasing exponentially with path length. Although Monte Carlo techniques are essentially impervious to the size of the state space (they also work with non-denumerable space), it is easy to construct verification problems for which there is a unique optimal scheduler. Such examples do not necessarily invalidate the approach, however, because it may not be necessary to find the possibly unique optimal scheduler to return a result with a level of statistical confidence. The nature of the distribution of schedulers nevertheless affects efficiency, so in this section we explore the convergence properties of smart sampling and give an example from the literature that does not converge as well as the case studies in Section 6.

Essentially, the problem is that of exponentially distributed schedulers, i.e., distributions having a very low mass of near optimal schedulers. Figure 10 illustrates the difference between exponentially decreasing and linearly decreasing distributions with the same overall mass. In both cases $p_{\max} \approx 0.2$ (the density at 0.2 is zero), but the figure shows that there is more probability mass near 0.2 in the case of the linear distribution.

Figure 9 illustrates the convergence of Algorithm 4, using a per-iteration budget of 10^6 applied to schedulers whose probability of success (i.e., of satisfying a hypothetical property) is distributed according to the exponential distribution of Fig. 10. Figure 9a shows how the initial undirected sampling (dots) can identify a crude approximation of $p_{\max}$. This approximation is then used to generate the candidate set of schedulers (distribution indicated with an arrow). The other lines illustrate five iterations of refinement, resulting in a shift of the distribution towards $p_{\max}$. Figure 9b illustrates the same shift in terms of the convergence of probability estimates. Iteration 0 corresponds to the uninformed sampling. Iteration 1 corresponds to the generation of the candidate set of schedulers. Note that for these first two iterations, $\hat{p}_{\text{mean}}$ includes schedulers that have zero probability of success. The expected value of $\hat{p}_{\text{mean}}$ is therefore equal to the total mass of non-zero probabilities in the distribution (≈ 0.0144), the expected probability of estimates produced by the uniform probabilistic scheduler. This fact can be used to verify that the hash function and PRNG described in Section 4 sample uniformly. In subsequent iterations the candidates all have non-zero probability of success. Importantly, the figure demonstrates that there is a significant increase in the maximum probability of scheduler success ($\sigma_{\max}$) between iteration 0 and iteration 1, and that this maximum is maintained throughout the subsequent refinements. Despite the apparently very low density of schedulers near $p_{\max}$, Algorithm 4 is able to make a good approximation.

The theoretical performance demonstrated in Fig. 9 explains why we are able to achieve good results in Section 6. It is nevertheless possible to find examples for which accurate results are difficult to achieve. Figure 11 illustrates the results of applying Algorithm 4



(a) Scheduler distributions. Dots are results of uninformed sampling. Arrow indicates initial candidate set. Other lines show results of subsequent refinements.

(b) Estimates and schedulers. At each iterative step: $\hat{p}_{\max}$ is the maximum estimate, $\hat{p}_{\text{mean}}$ is the mean estimate and $\sigma_{\max}$ is the true maximum probability of the available schedulers.

Fig. 9: Convergence of Algorithm 4 with exponentially distributed scheduler probabilities (Fig. 10) and per-iteration budget of 10^6 simulations.

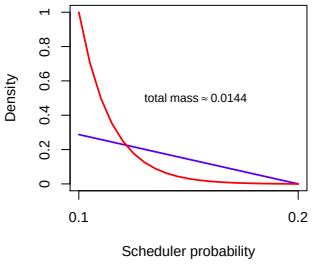


Fig. 10: Theoretical linear and exponential scheduler densities with probability mass ≈ 0.0144 and zero density at probability 0.2.

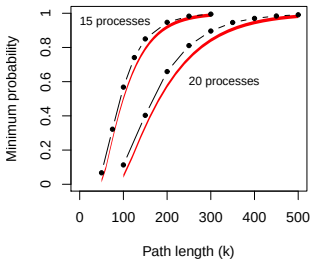


Fig. 11: Performance of smart sampling (dots) applied to self-stabilising models of [18]. Shaded areas denote true values ± 0.01.

to instances of the self-stabilising algorithm of [18], using a per-iteration budget of 10^5. Although the estimates (dots) do not lie within our statistical confidence bounds of the true values (shaded areas), we nevertheless make the claim that the results are useful. In general, given a Chernoff bound specified according to (9), our approach is able to provide extremal probability estimates for intractable MDPs, which are guaranteed not to be greater than the true maximum nor less than the true minimum by more than ε with probability δ.

To improve the performance of smart sampling, it is possible to make an even better allocation of simulation budget. For example, if good schedulers are very rare it may be beneficial to increase the per-iteration budget (thus increasing the possibility of seeing a good scheduler in the initial candidate set) but increase the proportion of schedulers rejected after each iteration (thus reducing the overall number of iterations and maintaining a fixed total number of simulations). To avoid rejecting good schedulers under such a regime, it may be necessary to

reject fewer schedulers in the early iterations when confidence is low.

8 Prospects and Challenges

The use of sampling facilitates algorithms that scale independently of the sample space, hence we anticipate that it will be possible to apply our techniques to nondeterministic models with non-denumerable schedulers. We believe it is immediately possible to apply smart sampling to reward-based MDP optimisation problems.

The success of sampling depends on the relative abundance of near optimal schedulers in scheduler space and our experiments suggest that these are not rare in standard case studies. While it is possible to construct pathological examples, where near optimal schedulers cannot easily be found by sampling, it is perhaps even simpler to confound numerical techniques with state explosion (three independent counters ranging over 0 to 1000 is typically sufficient with current hardware). Hence, as with numerical model checking, our ongoing challenge is essentially to increase performance and increase the number of models and problems that may be efficiently addressed. Smart sampling has made significant improvements over simple sampling, but we recognise that it will be necessary to develop other techniques to accelerate convergence. We anticipate that the most fruitful approaches will be (i) to reduce the sampled scheduler space to only those that satisfy the property and (ii) to construct schedulers piecewise. Such techniques will also reduce the potential of hash function collisions.

An important remaining challenge is to quantify the confidence of our estimates and hypothesis tests with respect to optimality. In the case of hypothesis tests that satisfy the hypothesis, the statistical confidence of the result is sufficient. If an hypothesis is not satisfied, however, the statistical confidence does not relate to whether there exists a scheduler to satisfy it. Likewise, the statistical confidence bounds of probability estimates imply nothing about how close they are to the true optima. We nevertheless know that our estimates of the extrema must lie within the true extrema or exceed them with the specified statistical confidence. This is already useful and a significant improvement over the results produced using the uniform probabilistic scheduler. In addition, given the number of simulations performed, we may at least quantify confidence with respect to the product $p_g p_{\overline{g}}$ (the rarity of near optimal schedulers times the average probability of the property with near optimal schedulers).

Acknowledgements

We are grateful to Benoît Delahaye for useful prior discussions. This work was partially supported by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS).

References

1. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
2. Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.
3. Andrea Bianco and Luca De Alfaro. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer, 1995.
4. Jonathan Bogdoll, Luis María Ferrer Fioriti, Arnd Hartmanns, and Holger Hermanns. Partial order methods for statistical model checking and simulation. In *Formal Techniques for Distributed Systems*, pages 59–74. Springer, 2011.
5. Benoît Boyer, Kevin Corre, Axel Legay, and Sean Sedwards. PLASMA-lab: A flexible, distributable statistical model checking library. In Kaustubh Joshi, Markus Siegle, Mariëlle Stoelinga, and PedroR. D'Argenio, editors, *Quantitative Evaluation of Systems*, volume 8054 of *LNCS*, pages 160–164. Springer, 2013.
6. Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Kwiatkowska, David Parker, and Mateusz Ujma. Verification of markov decision processes using learning algorithms. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis*, volume 8837 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2014.
7. E.M. Clarke, E. A. Emerson, and J. Sifakis. Model checking: algorithmic verification and debugging. *Commun. ACM*, 52(11):74–84, November 2009.
8. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.
9. Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In *Computer Aided Verification*, pages 27–39. Springer, 2003.
10. M. C. W. Geilen. On the construction of monitors for temporal logic properties. *Electronic Notes in Theoretical Computer Science*, 55(2):181–199, 2001.
11. Rob Gerth, Doron Peled, Moshe Y. Vardi Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *In Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.
12. D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In *Proceedings of 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pages 412–416. IEEE, Nov 2001.
13. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.
14. Arnd Hartmanns and Mark Timmer. On-the-fly confluence detection for statistical model checking. In *NASA Formal Methods*, pages 337–351. Springer, 2013.

15. David Henriques, Joao G. Martins, Paolo Zuliani, André Platzer, and Edmund M. Clarke. Statistical model checking for Markov decision processes. In *Quantitative Evaluation of Systems, 2012 Ninth International Conference on*, pages 84–93. IEEE, 2012.
16. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.
17. William George Horner. A new method of solving numerical equations of all orders, by continuous approximation. *Philosophical Transactions of the Royal Society of London*, 109:308–335, 1819.
18. A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizating mutual exclusion. In *Proc. 9th Annual ACM Symposium on Principles of Distributed Computing (PODC '90)*, pages 119–131. ACM New York, 1990.
19. Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.
20. Donald E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1998.
21. M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.
22. Marta Kwiatkowska, Gethin Norman, and David Parker. Analysis of a gossip protocol in PRISM. *SIGMETRICS Perform. Eval. Rev.*, 36(3):17–22, November 2008.
23. Marta Kwiatkowska, Gethin Norman, David Parker, and Jeremy Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29:33–78, August 2006.
24. Marta Kwiatkowska, Gethin Norman, David Parker, and Maria Grazia Vigliotti. Probabilistic mobile ambients. *Theoretical Computer Science*, 410(12-13):1272–1303, 2009.
25. Marta Z. Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In *Proc.* 2nd *Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 169–187. Springer, 2002.
26. Richard Lassaigne and Sylvain Peyronnet. Approximate planning and verification for large Markov decision processes. In *Proc.* 27th *Annual ACM Symposium on Applied Computing*, pages 1314–1319. ACM, 2012.
27. A. Legay, S. Sedwards, and L.-M. Traonouez. Scalable verification of Markov decision processes. In *4th Workshop on Formal Methods in the Development of Software (FMDS 2014)*, LNCS. Springer, 2014.
28. Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety*, volume 2. Springer, 1995.
29. U. Ndukwu and A. McIver. An expectation transformer approach to predicate abstraction and data independence for probabilistic programs. In *Proc. 8th Workshop on Quantitative Aspects of Programming Languages (QAPL'10)*, 2010.
30. Masashi Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics*, 10(1):29–35, 1958.
31. Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.
32. Abraham Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.
33. Douglas J. White. Real applications of Markov decision processes. *Interfaces*, 15(6):73–83, 1985.
34. Douglas J. White. Further real applications of Markov decision processes. *Interfaces*, 18(5):55–61, 1988.
35. Douglas J. White. A survey of applications of Markov decision processes. *Journal of the Operational Research Society*, 44(11):1073–1096, Nov. 1993.
36. H. L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon University, 2005.
37. Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Computer Aided Verification*, pages 223–235. Springer, 2002.