Smart Sampling for Lightweight Verification of Markov Decision Processes

Pedro d'Argenio, Axel Legay, Sean Sedwards, Louis-Marie Traonouez

Smart Sampling for Lightweight Verification of Markov Decision Processes

Pedro D'Argenio*, Axel Legay†, Sean Sedwards† and Louis-Marie Traonouez†
*Universidad Nacional de Córdoba, Argentina, and †Inria Rennes – Bretagne Atlantique, France

***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 lightweight techniques to sample directly from scheduler space, but finding optimal schedulers by simple sampling may be inefficient. Here we propose "smart" sampling algorithms that can make substantial improvements in performance.**

I. Introduction

Markov decision processes describe systems that interleave nondeterministic *actions* and probabilistic transitions. This model has proved useful in many real optimisation problems [29], [30], [31] and, in particular, may be used to represent concurrent probabilistic programs (see, e.g., [6], [4]). 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 or the probability that a system will satisfy a given property. In the latter case, it is usual to calculate the upper and lower bounds.

For example, consider the network of computational nodes depicted in Fig. 1 (relating to the case study in Section VI-D). 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 average 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 average 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 the 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.

Fig. 2 shows a typical fragment of an MDP. Its execution semantics are as follows. 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 $(s_1, s_2, s_3, s_4, \dots)$.

To calculate the expected probability of a sequence of states, as required by verification, it is necessary to define how the nondeterminism in the MDP will be resolved. In the literature

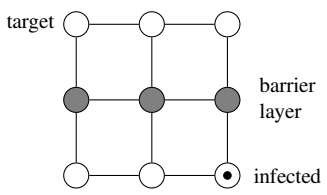


Figure 1: Model of network virus infection.

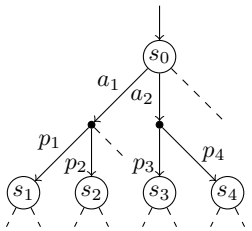


Figure 2: Fragment of a Markov decision process.

this is often called a *strategy*, a *policy*, an *adversary* or a *scheduler*. Classic analysis of MDPs is concerned with finding schedulers that maximise or minimise rewards assigned to each of the actions [5], [27]. In this work 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 [10]. *Probabilistic* model checking quantifies the probability that a probabilistic system will satisfy a property [12].

Numerical model checking algorithms to solve purely probabilistic systems are costly in time and space. Finding extremal schedulers in MDPs is generally more so, because it is effectively necessary to consider all possible ways that the nondeterminism might be resolved [6]. While memory-efficient ("lightweight") Mone Carlo techniques have been developed to address the probabilistic problem (i.e., *statistical* model checking), until recently [24] it has not been possible to address the nondeterministic problem in this way.

Building on the techniques introduced in [24], in this work we present "smart sampling" algorithms that make efficient use of a simulation budget and thus make significant improvements to lightweight verification of MDPs. We have implemented the algorithms in our statistical model checking platform, PLASMA [2], and demonstrate their performance on a number of case studies from the literature. Lightweight verification of MDPs is nevertheless in its infancy, so we also highlight counterexamples that motivate future work.

A. The Problems of Schedulers and State Explosion

The classic algorithms to solve MDPs are *policy iteration* and *value iteration* [27]. Model checking algorithms for MDPs may use value iteration applied to probabilities [4, Ch. 10] or solve the same problem using linear programming [6].

All consider *history-dependent* schedulers. Given an MDP with set of actions A, having a set of states S that induces a set of sequences of states $\Omega = S^+$, a history-dependent (general) scheduler is a function $\mathfrak{S} : \Omega \to A$. A memoryless scheduler is a function $\mathfrak{M} : S \to A$. Intuitively, at each state in the course of an execution, a general scheduler ($\mathfrak{S}$) chooses an action based on the sequence of previous states and a memoryless scheduler ($\mathfrak{M}$) chooses an action based only on the current state. History-dependent schedulers therefore include memoryless schedulers.

Fig. 3 illustrates a simple MDP for which memoryless and history-dependent schedulers give different optima for logical property $\mathbf{X}(\psi \wedge \mathbf{X}\mathbf{G}^t \neg\psi)$ when $p_1 \neq p_2$ and $t > 0$. The property makes use of the linear temporal operators *next* ($\mathbf{X}$) and *globally* ($\mathbf{G}$). 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 $t+1$ 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.

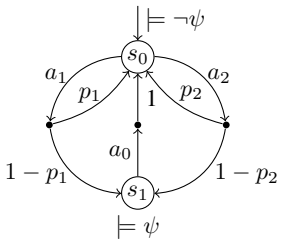


Figure 3: MDP with different optima for general and memoryless schedulers.

The principal challenge of finding optimal schedulers is what has been described as the 'curse of dimensionality' [5] and the 'state explosion problem' [10]: the number of states of a system increases exponentially with respect to the number of interacting components and state variables. This phenomenon has led to the design of sampling algorithms that find 'near optimal' schedulers to optimise rewards in discounted MDPs [17] and motivate our own work to develop lightweight verification algorithms to optimise probabilities.

The state explosion problem of model checking purely probabilistic systems has been well addressed by *statistical* model checking (SMC) [33]. SMC uses an executable model to approximate the probability that a system satisfies a specified property by the proportion of simulation traces that individually satisfy it. The state space of the system is not constructed explicitly – states are generated on the fly during simulation – hence SMC is 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.

SMC cannot be applied to MDPs without first resolving the nondeterminism. 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 (see Section II).

B. Our Approach

Using a hash function, we implement general schedulers by mapping the concatenation of an integer seed and a trace to a pseudo-randomly chosen action. By randomly sampling seeds, we are thus able to randomly sample schedulers. To find optimal schedulers we use "smart sampling" algorithms that make efficient use of a simulation budget. A small part of the budget is used to perform an initial assessment of the problem and to generate an optimal candidate set of schedulers. The remaining budget is used to test and refine the candidate set: sub-optimal schedulers are removed and their budget is reallocated to good ones.

C. Structure of the Paper

In Section II we briefly survey closely related work. In Section III we describe the elements of SMC that are relevant to the sequel. In Sections IV we introduce the basis of our lightweight verification techniques. In Section V we describe the notion of smart sampling and present our smart estimation and smart hypothesis testing algorithms. In Section VI we give the results of experiments with a number of case studies from the literature. In Section VII we discuss the limitations of smart sampling and in Section VIII we summarise the challenges and prospects for our approach.

II. Related Work

The Kearns algorithm [17] is the classic 'sparse sampling algorithm' for large, infinite horizon, discounted MDPs. It constructs a 'near optimal' scheduler piecewise, by approximating the best action from a current state, using a stochastic depth-first search. Importantly, optimality is with respect to rewards, not probability (as required by standard model checking tasks). 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, the algorithm recursively estimates the rewards of successive states, up to some maximum depth defined by the discount and desired error. Actions are enumerated while probabilistic choices are explored by sampling, with the number of samples set as a parameter. By iterating local exploration with probabilistic sampling, the discount guarantees that the algorithm eventually converges. The stopping criterion is when successive estimates differ by less than some error threshold.

There have been several recent attempts to apply SMC to nondeterministic models [7], [23], [14], [13], [24]. In [7], [13] 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 [23] the authors first find a memoryless scheduler that is near optimal with respect to a reward scheme and discount, 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 information about 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 rewards, not probability, hence [23] does not address the standard model checking problems of MDPs.

In [14] 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 given 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 approach has several problems. The inner loop does not in general converge to the true optimum, but is sometimes successful at finding an example 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.

Very recently, the elements of lightweight verification for MDPs were introduced in [24]. By sampling directly from history-dependent scheduler space using only $\mathcal{O}(1)$ memory, [24] opens up the possibility of scalable verification of MDPs. It is nevertheless easy to construct examples for which the simple sampling strategies of [24] are not feasible, motivating the present work.

III. Statistical Model Checking with Plasma

The algorithms we present here are implemented in our statistical model checking platform PLASMA (PLatform for Advanced Statistical Model checking Algorithms [8]). 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 features of SMC with PLASMA that are relevant to what follows.

SMC algorithms work by constructing an automaton to accept only traces that satisfy a specified property. The automaton may then be used to estimate the probability of the property or to decide an hypothesis about the probability. Typically, the probability of property φ is estimated by $\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}(\omega_i \models \varphi)$, where $\omega_1, \ldots, \omega_N$ are N independently generated simulation traces and $\mathbf{1}(\cdot)$ is an indicator function that corresponds to the output of the automaton: it returns 1 if the trace is accepted and 0 if it is not. As a statistical process, the results of SMC are given with probabilistic confidence.

In the case of estimation, PLASMA calculates a priori the required number of simulations according to a Chernoff bound [26] that allows the user to specify an error ε and a probability δ that the estimate $\hat{p}$ will not lie outside the true value $\pm\varepsilon$. Given that a system has true probability p of satisfying a property, the Chernoff bound ensures $\mathrm{P}(\mid \hat{p} - p \mid \geq \varepsilon) \leq \delta$. Parameter δ is related to the number of simulations N by $\delta = 2e^{-2N\varepsilon^2}$ [26], giving

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

In the case of hypothesis testing, PLASMA adopts the sequential probability ratio test (SPRT) of Wald [28] 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 (1) 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 [28]. 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

$$\mathit{ratio} = \prod_{i=1}^{n} \frac{(p^1)^{\mathbf{1}(\omega_i \models \varphi)}(1-p^1)^{\mathbf{1}(\omega_i \not\models \varphi)}}{(p^0)^{\mathbf{1}(\omega_i \models \varphi)}(1-p^0)^{\mathbf{1}(\omega_i \not\models \varphi)}},$$

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 [32]. 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 VI make use of the IGRIDA computing grid [1].

IV. Lightweight Verification of MDPs

Storing schedulers as explicit mappings does not scale, so we represent schedulers using uniform pseudo-random number

generators (PRNG) that are initialised by a *seed* and iterated to generate the next pseudo-random value. 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 schedulers cannot be identified, so it is not possible to estimate the probability of a property under a specific scheduler.

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 previous probabilistic choices. The i^{th} chosen action can be neither state nor trace dependent, as required by scheduler functions $\mathfrak{M}$ and $\mathfrak{S}$, respectively.

A. 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 schedulers functions $\mathfrak{S}$ and $\mathfrak{M}$ described in Section I-A. Importantly, the technique only relies on the standard properties of hash functions and PRNGs. 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

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

B. 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 [18, 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 [15][18, Ch. 4]: we set $h_0 = \sigma$ and find $h \equiv h_n$ (n as in Section IV-A) by iterating the recurrence relation

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

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 [11, Ch. 11]. Practically, it is an advantage to perform calculations using primitive data types that are native to the computational platform, so the sum in (2) 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:

$$(x + y) \bmod m \equiv (x \bmod m + y \bmod m) \bmod m \tag{3}$$

$$(xy) \bmod m \equiv ((x \bmod m)(y \bmod m)) \bmod m \tag{4}$$

The addition in (2) can thus be re-written in the form of (3), such that each term has a maximum value of $m - 1$:

$$h_i = ((h_{i-1} 2^{b_i}) \bmod m + (v_i) \bmod m) \bmod m \tag{5}$$

To prevent overflow, m must be no greater than half the maximum possible integer. Re-writing the first term of (5) in the form of (4), 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:

$$\begin{aligned}(h_{i-1}2^j) \bmod m &= (h_{i-1}2^{j-1}) \bmod m \\ &+ (h_{i-1}2^{j-1}) \bmod m \end{aligned} \quad (6)$$

Equation (6) allows our hash function to be implemented using efficient native arithmetic. Moreover, we infer from (2) 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.

C. Hypothesis Testing Multiple Schedulers

To decide whether there exists a scheduler such that $\mathrm{P}(\omega \models \varphi) \bowtie p$, we apply the SPRT to multiple (randomly chosen) schedulers. 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: Hypothesis testing 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: The result of the hypothesis test

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**
    $\omega \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma_i)$
    $ratio \leftarrow \frac{(p^1)^{\mathbf{1}(\omega \models \varphi)}(1-p^1)^{\mathbf{1}(\omega \not\models \varphi)}}{(p^0)^{\mathbf{1}(\omega \models \varphi)}(1-p^0)^{\mathbf{1}(\omega \not\models \varphi)}} ratio$
  if $ratio \leq A \wedge H = H_0 \vee ratio \geq B \wedge H = H_1$ **then**
    accept H

D. Estimating Multiple Schedulers

We consider the strategy of sampling M schedulers to estimate the optimum probability. We thus generate M estimates $\{\hat{p}_1, \ldots, \hat{p}_M\}$ 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 all estimates are 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 $N = \left\lceil -\ln\left(1 - \sqrt[M]{1-\delta}\right)/(2\varepsilon^2) \right\rceil$ for user-specified parameters M, ε and δ. 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 more 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

$$N = \left\lceil \left(\ln 2 - \ln\left(1 - \sqrt[M]{1-\delta}\right)\right)/(2\varepsilon^2) \right\rceil. \quad (7)$$

N scales logarithmically with M, making it tractable to consider many schedulers. In the case of $M = 1$, (7) degenerates to (1). Algorithm 3 is the resulting extremal probability estimation algorithm for multiple 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: Extremal estimates $\hat{p}_{\min}$ and $\hat{p}_{\max}$

Let $N = \left\lceil \ln(2/(1 - \sqrt[M]{1-\delta}))/(2\varepsilon^2) \right\rceil$ be the no. of simulations per scheduler
Let $\mathcal{U}_{\text{seed}}$ be a uniform PRNG and σ its sample
Initialise $\hat{p}_{\min} \leftarrow 1$ and $\hat{p}_{\max} \leftarrow 0$
Set seed of $\mathcal{U}_{\text{seed}}$ randomly
for $i \in \{1, \ldots, M\}$ **do**
  Iterate $\mathcal{U}_{\text{seed}}$ to generate σ_i
  Let $truecount = 0$ be the initial number of traces that satisfy φ
  for $j \in \{1, \ldots, N\}$ **do**
    $\omega_j \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma_i)$
    $truecount \leftarrow truecount + \mathbf{1}(\omega_j \models \varphi)$
  Let $\hat{p}_i = truecount/N$
  if $\hat{p}_{\max} < \hat{p}_i$ **then**
    $\hat{p}_{\max} = \hat{p}_i$
  if $\hat{p}_i > 0 \wedge \hat{p}_{\min} > \hat{p}_i$ **then**
    $\hat{p}_{\min} = \hat{p}_i$
if $\hat{p}_{\max} = 0$ **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

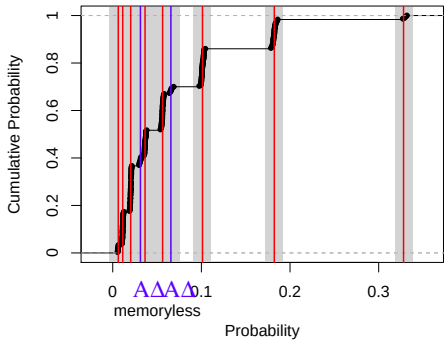


Figure 4: Empirical cumulative distribution of estimates from Algorithm 3 applied to MDP of Fig. 3.

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 red and blue lines mark the true probabilities of φ under each of the history-dependent and memoryless schedulers, respectively. The grey rectangles 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.

V. 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 optimal schedulers and the rarity of the property under an optimal scheduler. We should allocate our simulation budget accordingly and not waste budget on schedulers that are clearly not optimal.

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*).

A. Maximising the Probability of Seeing a Good Scheduler

In what follows we assume the existence of an MDP and a property φ whose probability we wish to maximise by choosing a suitable scheduler from the set $\mathfrak{S}$. Let $\mathcal{P} : \mathfrak{S} \to [0,1]$ be a function mapping schedulers to their probability of satisfying φ and let $p_{\max} = \max_{s\in\mathfrak{S}}(\mathcal{P}(s))$. For the sake of exposition, we consider the problem of finding a scheduler that maximises the probability of satisfying φ and define a "good" scheduler to be one in the set $\mathfrak{S}_g = \{s \in \mathfrak{S} \mid \mathcal{P}(s) \geq p_{\max} - \varepsilon\}$ 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 $p_{\min} = \min_{s\in\mathfrak{S}}(\mathcal{P}(s))$, 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.

Assuming 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 $p_{\overline{g}} = \sum_{s\in\mathfrak{S}_g} \mathcal{P}(s)/|\mathfrak{S}_g|$. If we select M schedulers uniformly 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

$$(1 - (1 - p_g)^M)(1 - (1 - p_{\overline{g}})^N). \tag{8}$$

Hence, for a given simulation budget $N_{\max} = NM$, to implement stage (*ii*) the idea is to choose N and M to maximise (8) 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 (8) using the heuristic $N = \lceil 1/p_{\overline{g}}\rceil$, which has near optimal performance in all but extreme cases.

B. 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. Lines 1 to 5 implement stage (*i*), lines 6 to 10 implement stage (*ii*) and lines 11 to 23 implement stage (*iii*). Note that the algorithm distinguishes $p_{\max}$ (the notional true maximum probability), $p_{\overline{\max}}$ (the true probability of the best candidate scheduler found) and $\hat{p}_{\overline{\max}}$ (the estimated probability of the best candidate scheduler).

The per-iteration simulation budget $N_{\max}$ must be greater than the number needed by the standard Chernoff bound (1), so ensure that there will at least be sufficient simulations to guarantee the specified confidence when 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 according to the Chernoff bound for multiple estimates (7), before refining the set of schedulers to a single element. Moreover, lines 16 to 19 allow the algorithm to quit as soon as the minimum number of simulations is reached.

C. Smart Hypothesis Testing

We wish to test the hypothesis that there exists a scheduler such that property φ has probability $\bowtie \theta$, where $\bowtie \in \{\geq, \leq\}$. Two advantages of sequential hypothesis testing are that it is not necessary to estimate the actual probability to know if

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 simulation budget / iteration

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

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 $\forall \sigma \in S, \forall i \in \{1, \ldots, N\} : \omega_i^\sigma \leftarrow \mathrm{Simulate}(\mathcal{M}, \varphi, \sigma)$
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 $\forall \sigma \in S, \forall i \in \{1, \ldots, N\} : \omega_i^\sigma \leftarrow \mathrm{Simulate}(\mathcal{M}, \varphi, \sigma)$
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 $\quad R' : \{1, \ldots, |S|\} \to S$ is an injective function s.t.
$\forall (n, \sigma), (n', \sigma') \in R',\ n > n' \implies R(\sigma) \geq R(\sigma')$
23 $\quad S \leftarrow \{\sigma \in S \mid \sigma = R'(n) \wedge n \in \{\lfloor |S|/2 \rfloor, \ldots, |S|\}\}$

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.

A sub-optimal approach would be to 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. Our approach is to exploit the fact that the *average* estimate at each iteration of Algorithm 4 is known with high confidence, i.e., confidence given by the total simulation budget. This follows directly from the result of [9], where the bound is specified for a sum of arbitrary random variables, not necessarily with identical expectations. By similar arguments based on [28], it follows that the sequential probability ratio test may also be applied to the sum of results produced during the course of an iteration of Algorithm 4. Moreover, it is possible to test each scheduler with respect to its individual results and the current number of schedulers, according to the bound given in Section IV-C.

Hence, 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 the "best" scheduler ever individually falsifies the hypothesis (lines 25 and 26), the algorithm also terminates and reports the result. Note that this outcome does not imply that there is no scheduler that will satisfy the hypothesis, only that no scheduler was found with the given budget. Finally, if the algorithm refines the initial set of schedulers to a single instance and the hypothesis was neither satisfied nor falsified, an inconclusive result is reported (line 29).

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 $N = \lceil 1/\theta \rceil$, (8) ensures that such schedulers, if they exist, have high probability of being observed. The initial coarse exploration 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.

VI. Case Studies

To demonstrate the performance of smart sampling, we implemented Algorithms 4 and 5 in our statistical model checking platform PLASMA [2], [8]. 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 we therefore 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.

A. IEEE 802.11 Wireless LAN Protocol

We consider a reachability property of the IEEE 802.11 Wireless LAN (WLAN) protocol model of [22]. The pro-

Algorithm 5: Smart Hypothesis Testing

Input:
$\mathcal{M}$: an MDP
φ: a property
$H : \mathrm{P}(\omega \models \varphi) \geq \theta \pm \varepsilon$ is the hypothesis
α, β: the desired error probabilities of H
$N_{\max}$: the per-step simulation budget
Output: The result of the hypothesis test

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 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 and quit
      if $ratio_i \geq B_M$ **then**
        Reject H (given budget) and quit
  $R' : \{1, \ldots, |S|\} \to S$ is an injective function s.t.
  $\forall (n, \sigma), (n', \sigma') \in R',\ n > n' \implies R(\sigma) \geq R(\sigma')$
  $S \leftarrow \{\sigma \in S \mid \sigma = R'(n) \wedge n \in \{\lfloor |S|/2 \rfloor, \ldots, |S|\}\}$
Inconclusive result (given budget)

tocol aims to avoid "collisions" between devices sharing a communication channel, by means of an exponential back-off procedure when a collision is detected. We therefore estimate the probability of the second collision at various time steps, using Algorithm 3 with per-iteration budget of 10^5 simulations. Fig. 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

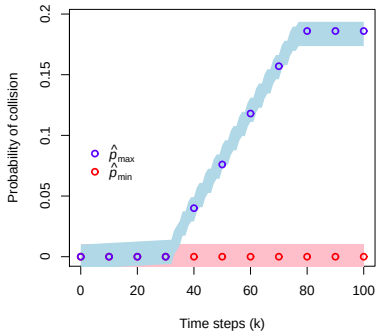


Figure 5: Estimated maximum and minimum probabilities of second collision in WLAN protocol. Shaded regions denote true values ± 0.01.

CSMA 3 4	θ	0.5	0.8	0.85	0.86	0.9	0.95
	Time (s)	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 (s)	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 (s)	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(s)	0.8	2.6	*	2.9	2.9	1.3
WLAN 6	Time(s)	1.3	2.2	*	6.5	1.3	1.3

Table I: Hypothesis test results for CSMA/CD and WLAN protocols. Time is simulation time to achieve the correct result on a single machine. Asterisks denote true probabilities.

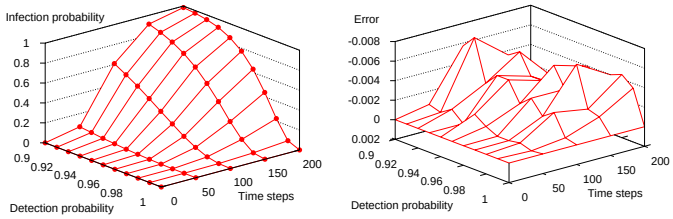
the true values. Table I gives the results of hypothesis tests based on the same model using property $\mathbf{F}^{100} col = 2$. See Section VI-B for a description.

B. 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 I we give the results of applying Algorithm 5 to the IEEE 802.3 CSMA/CD protocol model of [20]. The models and parameters are chosen to compare with results given in Table III in [14], hence we also give results for hypothesis tests performed on the WLAN model used in Section VI-A. In contrast to the results of [14], our results are produced on a single machine, with no parallelisation. There are insufficient details given about the experimental conditions in [14] 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.

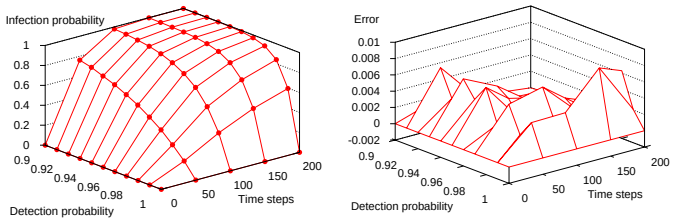
C. Choice Coordination

To demonstrate the scalability of our approach, we consider the choice coordination model of [25] 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



(a) Estimates. (b) Errors.

Figure 6: Minimum probability of network infection.



(a) Estimates. (b) Errors.

Figure 7: Maximum probability of network infection.

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.

D. 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 virus infection model of [3], based on [21]. 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. Figs. 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.

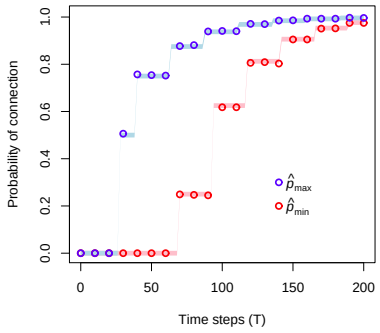


Figure 8: Estimated probabilities that maximum path length is < 4 in gossip protocol model. Shaded regions denote ± 0.01 of true values.

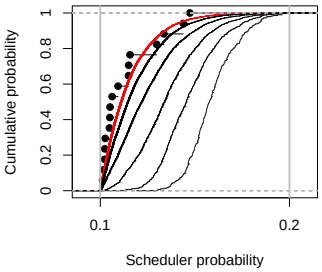
E. Gossip Protocol

Gossip protocols are an important class of network algorithms that rely on local connectivity to propagate information globally. Using the gossip protocol model of [19], 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 the property $\mathbf{F}^T \mathit{max\_path\_length} < 4$. 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.

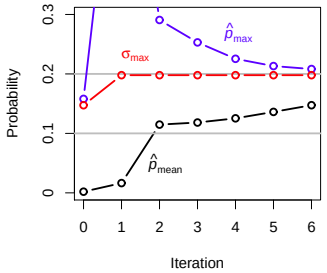
VII. Convergence and Counterexamples

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, give a counterexample from the literature and motivate our ongoing development of lightweight learning techniques.

Essentially, the problem is that of exponentially distributed schedulers, i.e., having a very low mass of schedulers close to the optimum. Fig. 10 illustrates the difference between



(a) Scheduler distributions. Dots denote the results of initial sampling. The red line is the set of schedulers to refine. Black lines show the result 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.

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

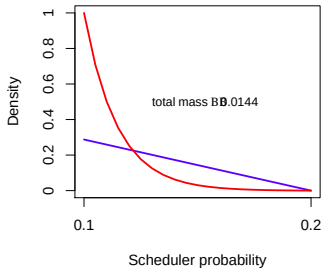


Figure 10: Theoretical linear (blue) and exponential (red) scheduler densities. Both have probability mass ≈ 0.0144 and zero density at scheduler probability 0.2.

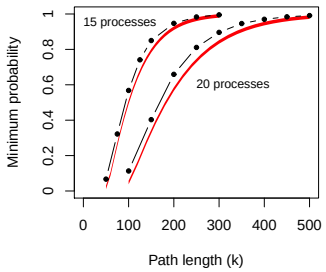


Figure 11: Performance of smart sampling (black dots) applied to self-stabilising models of [16]. Red shaded areas denote true values ± 0.01.

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. Fig. 9a shows how the initial undirected sampling (black dots) crudely approximates $p_{\max}$. This approximation is then used to generate the candidate set of schedulers (red distribution). The black lines illustrate five iterations of refinement, resulting in a shift of the distribution towards $p_{\max}$. Fig. 9b illustrates the same shift in terms of the convergence of probabilities. Iteration 0 corresponds to the undirected sampling. Iteration 1 corresponds to the generation of the candidate set of schedulers. For these first two iterations, $\hat{p}_{\text{mean}}$ includes the schedulers which have zero probability of success. 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 VI. It is nevertheless possible to find examples for which accurate results are difficult to achieve. Fig. 11 illustrates the results of applying Algorithm 4 to instances of the self-stabilising algorithm of [16], using a per-iteration budget of 10^5. We see that estimates (black dots) do not well approximate the true values (red shaded areas).

To improve the performance of our algorithms it is possible to better allocate 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.

Such "secondary optimisations" are unlikely to overcome problems that tend to scale exponentially. Hence, our main focus of future development will be lightweight learning algorithms that facilitate directed sampling and piecewise construction of schedulers.

VIII. Prospects and Challenges

We believe our techniques are immediately extensible to continuous time MDPs and other models that use nondeterminism. As noted in Section V-A, it is also possible to apply smart sampling to reward-based MDP optimisation problems. In this case a good scheduler would be one that produces an optimal reward and the initial candidate set of schedulers would be refined according to rewards, rather than probability.

By hashing the set of traces to a smaller set of hash codes, our algorithms sample from only a subset of all possible schedulers. This is not necessarily a problem, since any sampling approach will test far fewer schedulers than the maximum number of hash codes. In line with other statistical techniques, we are investigating means to quantify the confidence of chosen schedulers with respect to optimality.

Despite the foregoing, it is easy to construct examples where good schedulers are vanishingly rare. In such cases, merely quantifying the low confidence may not be useful. Our ongoing focus is therefore the development of lightweight learning techniques that construct schedulers piece-wise, to improve convergence and to consider a much larger set of schedulers.

Acknowledgement

This work was partially supported by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS).

REFERENCES

[1] IGRIDA coputing grid. igrida.gforge.inria.fr.

[2] PLASMA project web page. https://project.inria.fr/plasma-lab/.

[3] PRISM case studies. http://www.prismmodelchecker.org/casestudies/.

[4] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[5] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[6] 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.

[7] 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.

[8] 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 *Lecture Notes in Computer Science*, pages 160–164. Springer Berlin Heidelberg, 2013.

[9] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statist.*, 23(4):493–507, 1952.

[10] E.M. Clarke, E. A. Emerson, and J. Sifakis. Model checking: algorithmic verification and debugging. *Commun. ACM*, 52(11):74–84, November 2009.

[11] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[12] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[13] Arnd Hartmanns and Mark Timmer. On-the-fly confluence detection for statistical model checking. In *NASA Formal Methods*, pages 337–351. Springer, 2013.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Donald E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1998.

[19] Marta Kwiatkowska, Gethin Norman, and David Parker. Analysis of a gossip protocol in PRISM. *SIGMETRICS Perform. Eval. Rev.*, 36(3):17–22, November 2008.

[20] 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.

[21] Marta Kwiatkowska, Gethin Norman, David Parker, and Maria Grazia Vigliotti. Probabilistic mobile ambients. *Theoretical Computer Science*, 410(12-13):1272–1303, 2009.

[22] Marta Z. Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol. In *Proc. 2^{nd} Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 169–187. Springer-Verlag, 2002.

[23] Richard Lassaigne and Sylvain Peyronnet. Approximate planning and verification for large Markov decision processes. In *Proc. 27^{th} Annual ACM Symposium on Applied Computing*, pages 1314–1319. ACM, 2012.

[24] 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.

[25] 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.

[26] Masashi Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics*, 10(1):29–35, 1958.

[27] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.

[28] Abraham Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.

[29] Douglas J. White. Real applications of Markov decision processes. *Interfaces*, 15(6):73–83, 1985.

[30] Douglas J. White. Further real applications of Markov decision processes. *Interfaces*, 18(5):55–61, 1988.

[31] Douglas J. White. A survey of applications of Markov decision processes. *Journal of the Operational Research Society*, 44(11):1073–1096, Nov. 1993.

[32] H. L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon University, 2005.

[33] 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.