Estimating Rewards
Rare Events in Nondeterministic Systems

Axel Legay, Sean Sedwards, Louis-Marie Traonouez

Estimating Rewards & Rare Events in Nondeterministic Systems

Axel Legay, Sean Sedwards and Louis-Marie Traonouez

Inria Rennes – Bretagne Atlantique

Abstract. Exhaustive verification can quantify critical behaviour arising from concurrency in nondeterministic models. Rare events typically entail no additional challenge, but complex systems are generally intractable. Recent work on Markov decision processes allows the extremal probabilities of a property to be estimated using Monte Carlo techniques, offering the potential to handle much larger models. Here we present algorithms to estimate extremal rewards and consider the challenges posed by rarity. We find that rewards require a different interpretation of confidence and that reachability rewards require the introduction of an auxiliary hypothesis test. We show how importance sampling can significantly improve estimation when probabilities are low, but find it is not a panacea for rare schedulers.

1 Introduction

Complex systems often contain implicit elements of uncertainty, arising from human interactions or unknown deployment configurations. Such uncertainty may be expressed as nondeterminism in formal models and then analysed using formal verification. In this work we focus on Markov decision processes (MDP), although the techniques we present are adaptable to other formalisms that include nondeterminism.

MDPs interleave nondeterministic actions and probabilistic transitions and may be seen as comprising probabilistic subsystems whose transitions depend on the states of the other subsystems. Assigning rewards to actions, MDPs have proven useful in many real optimisation problems [20]. Rewards are a useful decoration to model quantitative aspects of systems that are not expressible as probabilities, such as information, energy and financial cost.

Given a system represented as an MDP, model checking [1] may be used to detect and optimise its critical performance, but exhaustive numerical algorithms have complexity related to the size of the system's state space and scale exponentially with respect to the number of independent variables in the model. The application of symmetry reductions and compositional approaches may help, but real systems are typically intractable due to their scale and heterogeneity.

Statistical model checking (SMC) describes a collection of Monte Carlo techniques that approximate the results of numerical model checking and aim to avoid considering the entire state space. States are generated on the fly during

simulation and results are given with statistical confidence. Monte Carlo techniques may be divided linearly on parallel computation architectures and are generally insensitive to the number of states of a model, but the number of simulations required for a given confidence scales quadratically with the rarity of the event being quantified. Since systems are designed to function correctly, failure is typically very rare and critically important. Rare event techniques, such as importance sampling [19] and importance splitting, have therefore been applied to SMC.

SMC requires a means to sample execution traces, but MDPs deliberately avoid completely specifying how a system executes. The notion of a scheduler is thus used to transform an MDP into a discrete time Markov chain that can be executed using stochastic simulation. The verification problem becomes one of finding optimal schedulers—schedulers that optimise some quantitative property.

Early attempts to apply SMC to MDPs store schedulers explicitly, but schedulers typically have the same complexity as the system as a whole, so these approaches tend to scale no better than numerical algorithms. In [15, 5] the authors use pseudo-random number generators to define the possibly infinite behaviour of schedulers in constant memory, allowing them to be sampled at random and tested individually. While sampling offers a significant computational saving over enumeration, it raises the problems of rare schedulers and rare events. The problem of rare schedulers has been partially ameliorated by 'smart sampling' [5], but properties with low probability make even non-rare schedulers difficult to find.

In this work we enhance the ideas of [15, 5] with importance sampling, to significantly increase the chance of finding optimal schedulers when properties have low probability. Since rewards properties have not yet been implemented in the sampling context of [15, 5], we also present an algorithm to find schedulers that approximately maximise or minimise expected rewards. Unlike probabilities, rewards have no inherent a priori bounds, so the standard statistical techniques to bound the absolute error do not apply. Our solution is to use a relative bound, based on a generalisation of the Chernoff-Hoeffding bound often used in SMC [17]. A further challenge is that the commonly used 'reachability rewards' [11] assume the probability of a property is known with absolute certainty and define the reward of properties with probability less than 1 to be infinite. With no additional information about the property, this definition induces an unknown distribution with potentially infinite variance. As such its properties cannot be directly estimated by sampling. Our solution is to introduce an auxiliary hypothesis test to assert that the probability of the property is 1. The estimation results can then be said to lie within the specified confidence bounds given that the hypothesis is true, while the confidence of the hypothesis test can be made arbitrarily high.

In Section 2 we provide the notational definitions used in the sequel. In Section 3 we review the notions of lightweight verification that underlie our approach. In Section 4 we briefly describe smart sampling [5] and show how it may be profitably extended with importance sampling. In Section 5 we present

our algorithm for rewards properties. In Section 7 we present experimental results that demonstrate the effectiveness and limitations of our algorithms. Section 8 concludes the paper.

Related Work

The Kearns algorithm [10] is a well known sampling algorithm for discounted MDPs which approximately optimises rewards over infinite horizons. The rewards used in model checking, however, are typically not discounted and are defined over finite horizons [11].

In [14] the authors use an adaptation of the Kearns algorithm to find a memoryless scheduler that is near optimal with respect to a discounted reward scheme. The resulting scheduler induces a Markov chain whose properties may be verified with standard SMC, but these are not rewards properties of the original MDP.

In [2, 6] the authors present algorithms to remove 'spurious' nondeterminism on the fly, so that standard SMC may be used. This approach is limited to the class of MDPs whose nondeterminism is not affected by scheduling.

In [7] the authors count the occurrence of state-actions in simulations, to iteratively improve a probabilistic scheduler that is assessed using sequential hypothesis testing. If an example that satisfies the hypothesis is found it is correct, but the frequency of state-actions is not in general indicative of global optimality.

In [3] the authors present learning algorithms to bound the maximum probability of (unbounded) reachability properties. The algorithms refine upper and lower bounds associated to state-actions, according to the contribution of individual simulations. The algorithms converge very slowly and may not converge to the global optimum.

In [15] the authors define the behaviour of a scheduler by combining a pseudo-random number generator with an incremental hash function. Schedulers are selected at random and SMC is applied to each of the discrete time Markov chains they induce, to find one that is approximately optimal. This simple sampling approach is made more efficient by 'smart sampling' in [5]: an initial candidate set of randomly selected schedulers is progressively refined by discarding those which are sub-optimal and re-assigning their simulation budget to the schedulers that remain. The present work builds on these results, which are reviewed in more detail in Sections 3 and 4.

Importance sampling has been widely applied to quantify failure in 'highly reliable' systems (e.g., [19]) and more recently in the specific context of SMC (e.g., [9]). We believe the present work is the first to apply importance sampling to MDPs.

2 Preliminaries

In this work 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 proceeds by a sequence of transitions between states, starting from an initial state, inducing a set of possible 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 $T(s,a,d)$. We assume that rewards are defined by some function $R : S^+ \to \mathbb{Q}$ or $R : A^+ \to \mathbb{Q}$ that maps a sequence of states or a sequence of actions to a total reward. In what follows we abuse the notation and simply write $R(\omega)$ to mean the total reward assigned to trace $\omega \in \Omega$ according to an arbitrary reward scheme.

Our algorithms find deterministic schedulers that approximately maximise or minimise expected rewards and probabilities for an MDP. A history-dependent 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 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. History-dependent schedulers therefore include memoryless schedulers. In the context of SMC we consider finite simulation traces of bounded length, hence $\mathfrak{S}$ and $\mathfrak{M}$ are finite.

A scheduler applied to an MDP induces a Markov chain over which there is a probabilistic measure $F : \Omega \to \mathbb{R}$ such that $\int_\Omega \mathrm{d}F = 1$. Given a set of paths $\omega \in \Omega$ that satisfy some bounded linear temporal logic property φ, denoted $\omega \models \varphi$, the probability of φ is given by $p = \int_\Omega \mathbf{1}(\omega \models \varphi)\mathrm{d}F$, where $\mathbf{1} : \{\mathit{true}, \mathit{false}\} \to \{0,1\}$ is an indicator function that returns 1 if its argument is *true* and 0 otherwise. To estimate p, SMC algorithms typically construct an automaton to decide the truth of the statement $\omega_i \models \varphi$, i.e., whether concrete simulation trace ω_i satisfies property φ. The estimated probability of φ is then given by

$$\hat{p} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\omega_i \models \varphi) \qquad \omega_i \sim F, \tag{1}$$

where $\omega_1, \ldots, \omega_N$ are N statistically independent random simulation traces distributed according to F, denoted $\omega_i \sim F$, and $\hat{p}$ denotes the estimate of p. To bound the estimation error it is common to use the "Chernoff" bound of [17]. The user specifies an absolute error ε and a probability δ to define the bound $\mathrm{P}(\mid \hat{p} - p \mid \geq \varepsilon) \leq \delta$. The bound is guaranteed if N satisfies the relation

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

3 Lightweight Verification of Markov Decision Processes

We recall here the lightweight verification techniques of [15] that underlie our approach.

Schedulers as Pseudo-Random Number Generators To avoid storing schedulers as explicit mappings, we construct schedulers on the fly using uniform pseudo-random number generators (PRNG) that are initialised by a seed and iterated to generate the next pseudo-random value. Our technique uses two independent PRNGs that respectively resolve probabilistic and nondeterministic choices. The first is used in the conventional way to make pseudo-random choices during a simulation experiment. The second PRNG is used to choose actions such that the choices are consistent between different simulations in the same experiment. Given multiple simulation experiments, the further role of the second PRNG is to range uniformly over all possible sets of choices. The seed of the second PRNG can be seen as the identifier of a specific scheduler.

To estimate the probability of a property under a scheduler, we generate multiple probabilistic simulation traces by fixing the seed of the PRNG for nondeterministic choices while choosing random seeds for the PRNG for probabilistic choices. To ensure that we sample from history-dependent schedulers, we construct a per-step PRNG seed that is a hash of a large integer representing the sequence of states up to the present and a specific scheduler identifier [15].

A Hash Code to Identify a Trace We assume that the state of an MDP is an assignment of values to a vector of n system variables $v_i, i \in \{1, \ldots, n\}$, with each v_i represented by a number of bits b_i. The state can thus be represented by the concatenation of the bits of the system variables, while a sequence of states (a trace) may be represented by the concatenation of the bits of all the states. 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 as the *trace vector*. A scheduler is identified by an integer σ of b_σ bits, 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. In this way we can approximate the scheduler functions $\mathfrak{S}$ and $\mathfrak{M}$: $\mathcal{H}$ 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 also deterministically dependent on the trace and the scheduler. Algorithm 1 implements these ideas as a simulation function that returns a trace, given a scheduler and bounded temporal property as input. The uniformity of scheduler selection is demonstrated by the accuracy of the estimates labelled 'uniform prob' in Figs. 1–3.

An Efficient Incremental Hash Function To implement our approach we use an efficient hash function that constructs seeds incrementally using standard precision mathematical operations. The function is based on modular division, such that $h = (\sigma : \overline{s}) \bmod m$, where m is a large prime not close to a power of 2 [4, Ch. 11]. Since $\overline{s}$ is typically very large, we use Horner's method [4, Ch. 30] to generate h: we set $h_0 = \sigma$ and find $h \equiv h_n$ (n as above) by iterating the recurrence relation

$$h_i = (h_{i-1} 2^{b_i} + v_i) \bmod m. \tag{3}$$

Algorithm 1: Simulate

Input:
$\mathcal{M}$: an MDP with initial state s_0
φ: a bounded temporal logic property
σ: an integer identifying a scheduler

Output:
ω: a simulation trace

Let $\mathcal{U}_{\text{prob}}, \mathcal{U}_{\text{nondet}}$ be uniform PRNGs with respective samples $r_{\text{pr}}, r_{\text{nd}}$
Let $\mathcal{H}$ be a hash function
Let s denote a state, initialised $s \leftarrow s_0$
Let ω denote a trace, initialised $\omega \leftarrow s$
Let $\overline{s}$ be the trace vector, initially empty
Select seed of $\mathcal{U}_{\text{prob}}$ randomly
while $\omega \models \varphi$ *is not decided* **do**
    $\overline{s} \leftarrow \overline{s} : s$
    Set seed of $\mathcal{U}_{\text{nondet}}$ to $\mathcal{H}(\sigma : \overline{s})$
    Iterate $\mathcal{U}_{\text{nondet}}$ to generate r_{nd} and use to resolve nondeterministic choice
    Iterate $\mathcal{U}_{\text{prob}}$ to generate r_{pr} and use to resolve probabilistic choice
    Set s to the next state
    $\omega \leftarrow \omega : s$

Equation (3) allows us to generate a hash code knowing only the current state and the hash code from the previous step. When considering memoryless schedulers we need only know the current state. Using suitable congruences [15], the following equation allows (3) to be implemented using efficient native arithmetic:

$$(h_{i-1}2^j) \bmod m = (h_{i-1}2^{j-1}) \bmod m + (h_{i-1}2^{j-1}) \bmod m$$

In a typical implementation on current hardware, a hash function and PRNG may span around 10^{19} schedulers. This is usually many orders of magnitude more than the number of schedulers sampled. There is no advantage in sampling from a larger set of schedulers until the number of samples drawn approaches the size of the sample space.

Confidence with Multiple Estimates To avoid the cumulative error when choosing a single probability estimate from a number of alternatives, [15] defines the following Chernoff bound for multiple estimates:

$$N \geq \left\lceil \left(\ln 2 - \ln \left(1 - \sqrt[M]{1-\delta} \right) \right) / (2\varepsilon^2) \right\rceil . \tag{4}$$

Given M estimates $\{\hat{p}_1, \ldots, \hat{p}_M\}$ of corresponding true probabilities $\{p_1, \ldots, p_M\}$ each generated with N samples, (4) asserts that for any estimate $\hat{p}_i$, in particular the minimum or maximum, $\mathrm{P}(\mid \hat{p}_i - p_i \mid \geq \varepsilon) \leq \delta$. Note that when $M = 1$, (4) degenerates to (2).

4 Smart Sampling with Importance Sampling

Smart sampling [5] builds on the foregoing techniques to maximise the probability of finding an optimal scheduler with a finite simulation budget. It works by iteratively eliminating sub-optimal schedulers from a candidate set and reallocating their budget to those that remain.

The problem of finding optimal schedulers by sampling has two independent components: the probability of near-optimal schedulers (denoted p_g) and the average probability of the property under near-optimal schedulers (denoted $p_{\overline{g}}$). A near-optimal scheduler is one whose reward or probability (depending on the context) is within some ε of the optimal value. If we select M schedulers at random and verify each with N simulations, the expected number of traces that satisfy the property using a near-optimal scheduler is thus $Mp_gNp_{\overline{g}}$. The probability of seeing a trace that satisfies the property using a near-optimal scheduler is the cumulative probability

$$(1 - (1 - p_g)^M)(1 - (1 - p_{\overline{g}})^N). \tag{5}$$

To maximise the chance of finding a good scheduler with a simulation budget of $N_{\max} = NM$, N and M should be chosen to maximise (5). Then, following a sampling experiment using these values, any scheduler that produces at least one trace that satisfies the property becomes a candidate for further investigation. Since the values of of p_g and $p_{\overline{g}}$ are usually unknown a priori, it is necessary to perform an initial uninformed sampling experiment to estimate them, setting $N = M = \lceil \sqrt{N_{\max}} \rceil$. The results can be used to numerically optimise (5), however an effective heuristic is to set $N = \lceil 1/\hat{p}_{\overline{g}} \rceil$, where $\hat{p}_{\overline{g}}$ is the maximum observed estimate (or minimum non-zero estimate in the case of finding minimising schedulers).

The best scheduler is found by iteratively refining the candidate set. At each iteration the per-iteration simulation budget ($N_{\max}$) is divided between the remaining candidates, simulations are performed and the average probability or reward for each scheduler is estimated. Schedulers whose estimates fall into the "worst" quantile (lower or upper half, depending on context) are discarded. Refinement continues until estimates are known with specified confidence, according to (4). With a per-iteration budget satisfying (2), the algorithm is guaranteed to terminate with a valid estimate.

Smart sampling offers a potentially exponential improvement in performance over the simple sampling strategy described in Sect. 3, but if the property is a rare event, i.e., $p_{\overline{g}}$ is small, finding a good scheduler may remain challenging even if schedulers are not rare. To address this we enhance the basic algorithm with the variance reduction technique of importance sampling, to make $p_{\overline{g}} \approx 1$ and thus maximise the number of schedulers considered.

Importance sampling is a standard variance reduction technique that works by weighting the executable model of a probabilistic system to make a rare event occur more frequently in simulations. The proportion of simulations in which the event occurs using the weighted model overestimates the true probability, but

the estimate may be exactly compensated by the weights. This arises from the equality $p = \int_\Omega \mathbf{1}(\omega \models \varphi)\mathrm{d}F = \int_\Omega \mathbf{1}(\omega \models \varphi)\frac{\mathrm{d}F}{\mathrm{d}G}\mathrm{d}G$, where F and G are measures over Ω such that $\mathbf{1}(\omega \models \varphi)\frac{\mathrm{d}F}{\mathrm{d}G} > 0, \forall \omega \models \varphi$. F is the original (unweighted) measure, G is the importance sampling (weighted) measure and $\frac{\mathrm{d}F}{\mathrm{d}G}$ is the so-called likelihood ratio. From this we construct the importance sampling counterpart of (1):

$$\hat{p} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(\omega_i \models \varphi)\frac{\mathrm{d}F(\omega_i)}{\mathrm{d}G(\omega_i)} \qquad \omega_i \sim G. \tag{6}$$

N simulation traces ω_i are generated under measure G and their contribution is compensated by the likelihood ratio, which is calculated on the fly. If G is constructed to strongly favour φ, a high percentage of traces will satisfy φ and (6) will converge much faster than (1).

We use this phenomenon to significantly improve the performance of smart sampling when $p_{\overline{g}} \ll 1$. If $p_{\overline{g}}$ denotes the fraction of traces we expect to satisfy φ using near-optimal schedulers under F and $p'_{\overline{g}}$ is the fraction of traces we expect to satisfy φ using near-optimal schedulers under G, then importance sampling will increase the expectation of seeing a near-optimal scheduler by $p'_{\overline{g}}/p_{\overline{g}}$. Typically, $p'_{\overline{g}} \approx 1$, so the improvement is $\approx 1/p_{\overline{g}}$.

The smart sampling algorithm with importance sampling follows the procedure described above, but substitutes $p'_{\overline{g}}, \hat{p}'_{\overline{g}}$ for $p_{\overline{g}}, \hat{p}_{\overline{g}}$ to optimise (5) and generate the initial candidate set of schedulers. Thereafter, the algorithm uses the corrected estimates, i.e., according to (6), to order and refine the candidate set.

Given F generated by an optimal scheduler, the theoretically optimal importance sampling distribution G^* is defined by $\mathrm{d}G^* = \mathbf{1}(\omega \models \varphi)\mathrm{d}F/p$, where p is the probability of φ. To maintain the advantages of sampling, G is typically generated by a syntactic re-parametrisation of F and constructed by efficient means that do not iterate over the entire state space, such as 'failure biasing' [19] and cross-entropy minimisation [9]. These techniques generally find G that only approximate G^*, but are nevertheless very effective. We make use of failure biasing in one of our case studies in Section 7 and leave cross-entropy minimisation for future work.

A well known limitation of importance sampling is that the standard statistical confidence bounds do not apply. In our application this is less important because the principal goal is to make near-optimal schedulers more visible. Once such schedulers have been found, we are at liberty to use other methods to estimate the probability of the property, such as standard Monte Carlo. The values estimated by importance sampling may nevertheless provide useful non-zero bounds when standard Monte Carlo returns a zero estimate because the probability is too low.

5 Statistical Model Checking of MDPs with Rewards

In the classic context, rewards are assigned to actions [18]. In the context of formal verification, rewards are often assigned to states or transitions between states [11]. In both cases the rewards are summed over the length of a trace and the expected reward is calculated by averaging the total reward with respect to the probability of the trace. In this work we focus on MDPs in the context of formal verification, but the mechanism of accumulating rewards is unimportant to our algorithms and we simply assume that a total reward is assigned to a finite trace.

The notions of probability estimation used in standard SMC can be adapted to estimate the expected reward of a trace. Given a function $R(\omega) \in [a, b]$, a, b finite, that assigns a total reward to simulation trace ω, the expected reward may be estimated by $\frac{1}{N}\sum_{i=1}^{N} R(\omega_i)$, where $\omega_1, \ldots, \omega_N$ are statistically independent simulation traces. Since rewards may take values outside $[0, 1]$, we must use Hoeffding's generalisation of (2) to bound the errors [8]. To guarantee $\mathrm{P}(\mid \hat{r} - r \mid \geq \varepsilon) \leq \delta$, where r and $\hat{r}$ are respectively the true and estimated values of expected reward, N is required to satisfy the relation

$$N \geq \left\lceil \ln(2/\delta) \times (a-b)^2/(2\varepsilon^2) \right\rceil . \tag{7}$$

For non-trivial problems the values of a and b are usually not known, while guaranteed a priori bounds (e.g., by assuming maximum or minimum possible rewards on each step) may be too conservative to be useful. Although it is possible to develop a strategy using a posteriori estimates of a and b, i.e., based on $\max_{i\in\{1,\ldots,N\}}(R(\omega_i))$ and $\min_{i\in\{1,\ldots,N\}}(R(\omega_i))$, we see that N depends on the ratio of the absolute error ε to the range of values $(a-b)$. The confidence of estimates of rewards may therefore be specified a priori as a percentage of the maximum range of the support of R. We adopt this idea in Algorithm 2, where we use (2) and (4) and assume that ε expresses a percentage as a fraction of 1.

Rewards Properties

The rewards properties commonly used in numerical model checking are based on an extension of the logic PCTL [11]. This extension defines 'instantaneous' rewards (the average reward assigned to the k^{th} state of all traces, denoted $\mathbf{I}^k$), 'cumulative' rewards (the average total reward accumulated up to the k^{th} state of all traces, denoted $\mathbf{C}^k$) and 'reachability' rewards (the average accumulated reward of traces that eventually satisfy property φ, denoted $\mathbf{F}\varphi$). Instantaneous and cumulative rewards are based on finite traces and can be immediately approximated by sampling, using (2) and (4) to bound the errors. Reachability rewards are based on unbounded reachability ($\mathbf{F}$) and require additional consideration.

By the definition of reachability rewards [11], properties that are not satisfied with probability 1 are assigned infinite reward. The rationale behind this is that if $\mathrm{P}(\mathbf{F}\varphi) < 1$, there must exist an infinite path that does not satisfy φ, whose

rewards will accumulate infinitely. This definition is somewhat arbitrary, since rewards are not constrained to be positive nor to have a minimum value. An infinite sum of positive and negative values can equate to zero and it is also possible for an infinite sum of positive values to converge, as in the case of discounted rewards.

The definition of reachability rewards makes sense in the context of numerical model checking, where paths are not considered explicitly and unbounded properties can be quantified with certainty, but it causes problems for sampling. In particular, using sampling alone it is not possible to say with certainty whether $\mathrm{P}(\mathbf{F}\varphi) = 1$, even if every observed trace of finitely many satisfies φ. Without additional guarantees, the random variable from which samples are drawn could include the value infinity, giving it infinite variance. Statistical error bounds, which generally rely on an underlying assumption of finite variance, will therefore not be correct without additional measures.

To accommodate the standard definition of reachability rewards, our solution is to implement $\mathbf{F}\varphi$ as $\mathbf{F}^k\varphi$, i.e., bounded reachability where φ must be true within k steps, with an auxiliary hypothesis test to assert that $\mathrm{P}(\mathbf{F}\varphi) = 1$ is true. A positive result is thus an estimate within user-specified confidence plus an accepted hypothesis within other user-specified confidence. A negative result is a similar estimate, but with an hypothesis that is not accepted. This approach is consistent with intuition and with the SMC ethos to provide results within statistical confidence bounds. The hypothesis test may be implemented in any number of standard ways. Our implementation uses a convenient normal approximation model, which we describe in Section 6.

In practice, the bound k for reachability rewards is set much longer than it is supposed necessary to satisfy φ and the hypothesis is of the form $\mathrm{P}(\mathbf{F}^k\varphi) \geq p_0$, $p_0 \lessapprox 1$. Intuitively, the more traces of length $\leq k$ that satisfy φ, the more confident we are that $\mathrm{P}(\mathbf{F}\varphi) \geq p_0$ is true. Traces that fail to satisfy φ after k steps may nevertheless satisfy φ if allowed to continue, hence the value of p_0 defines how certain we wish to be after k steps. If the hypothesis is rejected, we may either conclude that the average reward is infinite (by definition), accept the calculated average reward as a lower bound or increase k and try again.

Our SMC engine quits as soon as a property is satisfied or falsified, so there is very little penalty in setting k large when we require high confidence, i.e., when $p_0 \approx 1$. Simulations that satisfy the property will only take as many steps as necessary, independent of a much larger value of k, while those that do not satisfy the property will be few because the auxiliary hypothesis is falsified quickly when p_0 is close to 1.

For the standard rewards properties described above, the value of $p_{\overline{g}}$ in (5) is effectively 1. In the case of instantaneous and cumulative rewards, traces are not filtered with respect to a property, so the probability of acceptance is 1. In the case of reachability rewards, either nearly all traces satisfy the property ('nearly' because the auxiliary hypothesis test allows for the case that not all traces satisfy the property) or the reward is assumed to be infinite. Hence, the case of probabilities significantly less than 1 does not have to be quantified, just

detected. The consequence of this is that there is no need for an undirected simulation experiment to estimate $p_{\overline{g}}$ and the initial candidate set will contain the maximum number of schedulers for the specified budget, i.e., $N = 1$ and $M = N_{\max}$.

6 Smart Reward Estimation Algorithm

Algorithm 2 builds on Algorithm 4 in [5] to find schedulers that maximise rewards. The algorithm to minimise rewards follows intuitively: replace instances of 'max' with 'min' in lines 16, 17, 21 and the Output line, and replace line 20 with $S \leftarrow \{\sigma \in S \mid \sigma = Q'(n) \wedge n \in \{1, \ldots, \lceil |S|/2 \rceil\}\}$.

The reward property ρ may be of type instantaneous, cumulative or reachability, which are denoted $\mathbf{I}^k\varphi$, $\mathbf{C}^k\varphi$ and $\mathbf{F}^k\varphi$, respectively, to unify the description. The reward function $\mathcal{R}_\rho : \mathbb{N} \times \Omega \to \mathbb{Q}$ maps the identifier of a scheduler and a trace to a reward, given reward property ρ. In the case of $\mathbf{I}^k\varphi$ and $\mathbf{C}^k\varphi$, k is the standard user-specified parameter for these rewards and φ is implicitly $\mathbf{G}^k\mathit{true}$ (i.e., *true* in the initial state and for k steps). In the case of $\mathbf{F}^k\varphi$, φ is user-specified and k is set as large as feasible to satisfy the hypothesis $\mathrm{P}(\mathbf{F}^k\varphi) \geq p_0$, with confidence defined by α (described below). Given that our actual requirement is that $\mathrm{P}(\mathbf{F}\varphi) = 1$, both p_0 and α will typically be close to 1, such that very few traces will be necessary to falsify the hypothesis.

The initial candidate set of schedulers and corresponding estimates are generated in lines 1 to 4. Applying (5), 1 simulation is performed using each of $N_{\max}$ schedulers chosen at random. The function Q maps schedulers to their current estimate. A number of initialisations take place in lines 5 to 6.

The function *trues* is used by the auxiliary hypothesis test and counts the total number of traces per scheduler that satisfy the property. The variable *samples* is also used by the auxiliary hypothesis test and counts the total number of traces used per scheduler. The value of *conf*, initialised to 1 to ensure at least one iteration, is the probability that the estimates exceed their specified bounds (defined by ε), given the current number of simulations. The main loop (lines 6 to 20) terminates when *conf* is less than or equal to the specified probability δ. Typically, the per-iteration budget will be such that the required confidence is reached according to (4) before the candidate set is reduced to a single element. Lines 10 to 14 contain the main simulation loop, which quits as soon as the required confidence is reached. Lines 15 to 20 order the results by estimated reward and select the upper quantile of schedulers.

The auxiliary hypothesis test necessary for reachability rewards is provided in lines 21 to 23. To test $\mathrm{P}(\mathbf{F}^k\varphi) \geq p_0$, it considers the error statistic $Z = \mathit{samples} \times (\hat{p}_\varphi - p_0)/\sqrt{\mathit{samples} \times p_0(1-p_0)}$, where $\hat{p}_\varphi = \mathit{trues}(\sigma_{\max})/\mathit{samples}$ is the estimate of $\mathrm{P}(\mathbf{F}^k\varphi)$. For typical values of *samples*, the distribution of Z is well approximated by a normal with mean = 0 and variance = 1 when the expectation of $\hat{p}_\varphi$, denoted $\mathrm{E}(\hat{p}_\varphi)$, is equal to p_0. To test the hypothesis with confidence α, the algorithm compares the statistic Z with the standard normal quantile of order α, denoted $z(\alpha)$. The value of $z(\alpha)$ may be drawn from a table

Algorithm 2: Reward Estimation

Input:

$\mathcal{M}$: an MDP

$\rho \in \{\mathbf{I}^k\varphi, \mathbf{C}^k\varphi, \mathbf{F}^k\varphi\}$: a reward property

$\mathcal{R}_\rho$: the reward function for ρ

$H_0 : \mathrm{P}(\mathbf{F}^k\varphi) \geq p_0$: the auxiliary hypothesis

$z(\alpha)$: confidence of H_0, the normal quantile of order α

ε, δ: the reward estimation Chernoff bound

$N_{\max} > \ln(2/\delta)/(2\varepsilon^2)$: the per-iteration budget

Output:

$\hat{r}_{\overline{\max}} \approx r_{\max}$, where $r_{\overline{\max}} \approx r_{\max}$ and $\mathrm{P}(|r_{\overline{\max}} - \hat{r}_{\overline{\max}}| \geq \varepsilon) \leq \delta$

$N \leftarrow 1$, $M \leftarrow N_{\max}$

$S \leftarrow \{M \text{ seeds chosen uniformly at random}\}$

$\forall \sigma \in S, \forall j \in \{1, \ldots, N\} : \omega_j^\sigma \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$

$Q \leftarrow \{(\sigma, q) \mid \sigma \in S \wedge \mathbb{Q} \ni q = \sum_{j=1}^{N} \mathcal{R}_\rho(\sigma, \omega_j^\sigma)/N\}$

$\forall \sigma \in S : \mathit{trues}(\sigma) \leftarrow 0$

$\mathit{samples} \leftarrow 0$, $\mathit{conf} \leftarrow 1$, $i \leftarrow 0$

while $\mathit{conf} > \delta \wedge S \neq \emptyset$ **do**

  $i \leftarrow i + 1$

  $M_i \leftarrow |S|$, $N_i \leftarrow 0$

  while $\mathit{conf} > \delta \wedge N_i < \lceil N_{\max}/M_i \rceil$ **do**

    $N_i \leftarrow N_i + 1$

    $\mathit{conf} \leftarrow 1 - (1 - e^{-2\epsilon^2 N_i})^{M_i}$

    $\forall \sigma \in S : \omega_{N_i}^\sigma \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$

    $\mathit{samples} \leftarrow \mathit{samples} + 1$

  $Q \leftarrow \{(\sigma, q) \mid \sigma \in S \wedge \mathbb{Q} \ni q = \sum_{j=1}^{N_i} \mathcal{R}_\rho(\sigma, \omega_j^\sigma)/N_i\}$

  $\sigma_{\max} \leftarrow \arg\max_{\sigma \in S} Q(\sigma)$

  $\hat{r}_{\overline{\max}} \leftarrow Q(\sigma_{\max})$

  $\forall \sigma \in S, j \in \{1, \ldots, N_i\} : \mathit{trues}(\sigma) = \mathit{trues}(\sigma) + \mathbf{1}(\omega_j^\sigma \models \varphi)$

  $Q' : \{1, \ldots, |S|\} \to S$ is an injective function s.t.
  $\forall (n, \sigma), (n', \sigma') \in Q' : n > n' \implies Q(\sigma) \geq Q(\sigma')$

  $S \leftarrow \{\sigma \in S \mid \sigma = Q'(n) \wedge n \in \{\lfloor |S|/2 \rfloor, \ldots, |S|\}\}$

$Z \leftarrow (\mathit{trues}(\sigma_{\max}) - \mathit{samples} \times p_0)/\sqrt{\mathit{samples} \times p_0\,(1 - p_0)}$

if $Z \leq z(\alpha)$ **then**

  H_0 is rejected

or approximated numerically. If $\mathrm{E}(\hat{p}_\varphi) \geq p_0$, the value of Z will be $\geq z(\alpha)$ with probability $\geq \alpha$.

To simplify the presentation, the auxiliary hypothesis test is also used by the instantaneous and cumulative rewards. In these latter cases, however, the test is guaranteed to always be satisfied.

7 Case Studies

The following results demonstrate typical performance on a selection of standard case studies. We necessarily use models whose expected rewards can be calculated or inferred using numerical methods, but observe that this does not give our algorithms any advantage. The models and properties can be found on the PLASMA website[1] and are illustrated in detail on the PRISM case studies website[2]. All timings are based on parallel simulations using 64 simulation cores of the IGRIDA cluster[3].

In most instances of reward estimation we were able to achieve accurate results with a relatively modest per-iteration simulation budget of $N_{\max} = 10^5$ simulations, using a Chernoff bound of $\varepsilon = \delta = 0.01$. In the case of the gossip protocol this budget was apparently not sufficient for all considered parameters. We nevertheless claim that the results provide useful conservative bounds. Note that for all reachability rewards we made the value of k in the auxiliary hypothesis test sufficiently large to ensure that all traces satisfied the property, giving us maximum confidence for the specified budget.

The figures plot the estimates generated by our algorithms against the values calculated using numerical methods. In addition to maximum and minimum, we also plot the the average estimates of the initial candidate set. This average corresponds to the expected reward or probability using the uniform probabilistic scheduler or, equivalently, treating the MDP as a DTMC with uniform probabilities assigned to nondeterministic choices. The average estimates are consistently accurate, demonstrating that Algorithm 1 is sampling uniformly from scheduler space.

Network Virus Infection This case study is based on [13] and comprises a network of linked sets of nodes. Initially, there is a set containing one node infected by a virus, a set with no infected nodes and a set of uninfected barrier nodes that divides the first two sets. A virus chooses which node to infect nondeterministically and infects it with probability 0.5. A node detects a virus probabilistically. Figure 1 plots the results of using a rewards property to estimate the expected number of attacks before a particular node is infected, varying the probability that a barrier node detects a virus as a parameter. Each point required approximately 15 seconds of simulation time. All estimates are within $\pm 1\%$ of the true values. Figure 2 plots the results of using importance sampling to estimate the

[1] project.inria.fr/plasma-lab
[2] www.prismmodelchecker.org/casestudies
[3] igrida.gforge.inria.fr

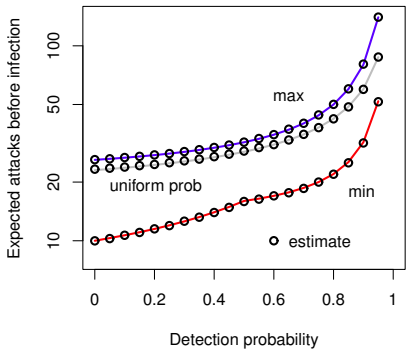

Fig. 1. Network virus infection: reward estimation.

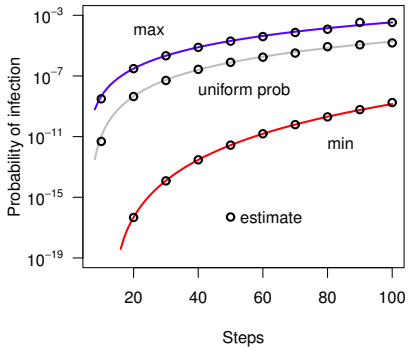

Fig. 2. Network virus infection: probability estimation using importance sampling.

probability of infection when the probability of a virus infecting a node is reduced to 0.01. Our importance sampling distribution is generated by the simple expedient of setting this parameter to its original value of 0.5. The plotted estimates are the average of 10 individual estimates generated using a per-iteration budget of just 10^4 simulations. In most cases this budget is orders of magnitude less than the expected number of simulations necessary to see a single trace that satisfies the property.

Gossip Protocol The gossip protocol of [12] uses local connectivity to propagate information globally. Using a reachability reward property, our algorithms accurately estimate the expected minimum and maximum number of rounds necessary for the network to become connected as 1.486 and 4.5, compared to correct values 1.5 and 4.5. The average simulation time per estimate was approximately 1 minute.

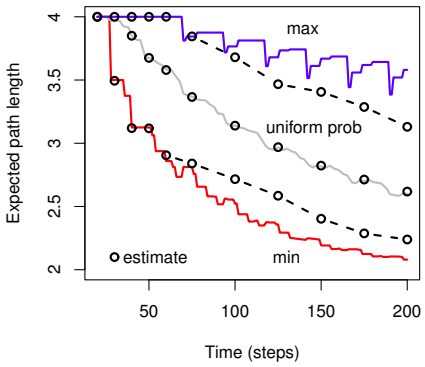

Fig. 3. Gossip protocol.

Figure 3 plots the maximum and minimum estimated path length at different time steps, using an instantaneous reward property. We see that the estimates are accurate up to about 75 time steps, but less so above this value. Since $p_{\overline{g}}$ in (5) is effectively 1, this problem cannot be overcome by importance sampling. The estimates are nevertheless guaranteed by (4) not to exceed the true values by more than a factor of $1 + \varepsilon$ with probability δ.

Choice Coordination To demonstrate the scalability of our approach we consider instances of the choice coordination model of [16] with $BOUND = 100$. This value makes most of the models intractable to numerical model checking, however it is possible to infer the correct values of rewards from tractable instances. The chosen reachability property gives the expected minimum number of rounds necessary for a group of tourists to meet. The following table summarises the results:

Number of tourists	2	3	4	5	6	7	8	9	10
Estimated minimum rounds to converge	4.0	5.0	7.0	8.0	10.0	11.0	12.0	13.0	14.0

All the estimates are exactly correct, while the average time to generate each result was just 8 seconds.

8 Prospects and Challenges

In this work we have focused on estimating the expected values of rewards and rare events. We believe the same techniques may be immediately extended to sequential hypothesis testing, as in [15] and [5]. Ongoing work suggests that estimating rewards in continuous time models will also be feasible.

With respect to rewards, our case studies demonstrate that our approach is effective and can be efficient with state space that is intractable to numerical methods. We do not yet provide numerical confidence with respect to optimality, but our techniques generate useful conservative bounds with correct statistical guarantees of accuracy: the estimate will be greater (less) than the true maximum (minimum) expected reward by a factor of $\geq 1 + \varepsilon$ with probability $\leq \delta$.

The use of importance sampling can improve the performance of smart sampling by a factor of $1/p_{\overline{g}}$. We have demonstrated this technique using an importance sampling distribution generated by simple failure biasing, but as future work we propose to develop a cross-entropy minimisation algorithm to find an optimal parametrised distribution, along the lines of [9].

In the case of standard rewards properties, $p_{\overline{g}} = 1$ and importance sampling cannot help. Figure 3 illustrates circumstances where our chosen per-iteration budget of 10^5 is apparently not sufficient to explore the tails of the distribution of schedulers. Merely increasing the budget will not in general be adequate, since near-optimal schedulers may be arbitrarily rare. Our proposed future solution is to combine sampling with learning to construct composite schedulers.

Acknowledgements

This work was partially supported by the European Union Seventh Framework Programme under grant agreement numbers 295261 (MEALS) and 318490 (SENSATION).

References

1. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
2. 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.
3. 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-Franois Raskin, editors, *Automated Technology for Verification and Analysis*, volume 8837 of *LNCS*, pages 98–114. Springer, 2014.
4. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3^{rd} edition, 2009.
5. Pedro D'Argenio, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Smart sampling for lightweight verification of Markov decision processes. *International Journal on Software Tools for Technology Transfer*, 17(4):469–484, 2015.
6. Arnd Hartmanns and Mark Timmer. On-the-fly confluence detection for statistical model checking. In *NASA Formal Methods*, pages 337–351. Springer, 2013.
7. David Henriques, Joao G. Martins, Paolo Zuliani, André Platzer, and Edmund M. Clarke. Statistical model checking for Markov decision processes. In 9^{th} *International Conference on Quantitative Evaluation of Systems (QEST2012)*, pages 84–93. IEEE, 2012.
8. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.
9. Cyrille Jegourel, Axel Legay, and Sean Sedwards. Cross-entropy optimisation of importance sampling parameters for statistical model checking. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification*, volume 7358 of *LNCS*, pages 327–342. Springer, 2012.
10. 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.
11. 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.
12. Marta Kwiatkowska, Gethin Norman, and David Parker. Analysis of a gossip protocol in PRISM. *SIGMETRICS Perform. Eval. Rev.*, 36(3):17–22, November 2008.
13. Marta Kwiatkowska, Gethin Norman, David Parker, and Maria Grazia Vigliotti. Probabilistic mobile ambients. *Theoretical Computer Science*, 410(12-13):1272–1303, 2009.
14. 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.
15. A. Legay, S. Sedwards, and L.-M. Traonouez. Scalable verification of Markov decision processes. In 4^{th} *Workshop on Formal Methods in the Development of Software (FMDS 2014)*, LNCS. Springer, 2014.
16. U. Ndukwu and A. McIver. An expectation transformer approach to predicate abstraction and data independence for probabilistic programs. In *Proc.* 8^{th} *Workshop on Quantitative Aspects of Programming Languages (QAPL'10)*, 2010.

17. Masashi Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics*, 10(1):29–35, 1958.
18. Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.
19. Perwez Shahabuddin. Importance sampling for the simulation of highly reliable Markovian systems. *Management Science*, 40(3):333–352, 1994.
20. Douglas J. White. A survey of applications of Markov decision processes. *Journal of the Operational Research Society*, 44(11):1073–1096, Nov 1993.