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

Approximate Planning and Verification for Large Markov Decision Processes

Richard Lassaigne, Sylvain Peyronnet

lassaigne@math.univ-paris-diderot.fr
Univ Paris VII, Logique Mathématique (IMJ), F-75251; CNRS, Paris-Centre, F-75000
sylvain.peyronnet@unicaen.fr
GREYC, Université de Caen Basse-Normandie, Caen, France

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

Abstract. We focus on the planning and verification problems for very large probabilistic systems, such as Markov Decision Processes (MDPs), from a complexity point of view. More precisely, we deal with the problem of designing an efficient approximation method to compute a near-optimal policy for the planning problem in discounted MDPs and the satisfaction probabilities of interesting properties, like reachability or safety, over the Markov chain obtained by restricting the MDP to the near-optimal policy.

In this paper we present two different approaches. The first one is based on sparse sampling while the second uses a variant of the multiplicative weigths update algorithm. The complexity of the first approximation method is independent of the size of the state space and uses only a probabilistic generator of the MDP. We give a complete analysis of this approach, for which the control parameter is mainly the targeted quality of the approximation. The second approach is more prospective and is different in the sense that the method is controlled by its speed of convergence.

Parts of this paper have already been presented in [24], by the same authors.

1 Introduction

Markov Decision Processes (MDPs) provide a powerful framework for modelling situations where a single controller needs to make decisions to achieve a certain goal under uncertainty. For example, MDPs allow to model control problems for communication systems, embedded systems and industrial software-based control systems. In this context, at each time unit, a decision maker or a controller observes the state of the system and chooses an action. The action choice has two consequences: the controller receives an immediate reward, or incurs an immediate cost, and the system evolves to a new state according to a probability distribution determined by the action choice. A policy provides the decision maker with a prescription for choosing an action in any possible future state.

For systems represented as MDPs, there are two different important problems. The first one is the planning problem: can we compute a policy which is optimal with respect to the sequence of rewards or costs? The second one is the verification problem: check if the model satisfies a specification expressed in a temporal logic framework or by a regular language. Verification frequently uses rewards/costs and, conversely, there are various approaches to planning based on temporal logic. Discounted MDPs are not a commonly used model in verification, while in planning they are widely used. The most studied approach for solving the planning problem in discounted MDPs is dynamic programming [19].

In this paper, we study these two problems from a complexity point of view. For very large models, as it is often the case for realistic applications, classical methods that use explicit or even abstract representation of models, become infeasible, due to the state space explosion phenomenon.

More precisely, we deal with the problem of designing efficient methods to approximate an optimal policy and the satisfaction probabilities of interesting properties in the model obtained by restricting the system to a near-optimal policy. Classical methods for the planning of discounted MDPs, such as value iteration and policy iteration [26], are known for being strongly polynomial (see the recent result of [13]), but are still intractable in the case of large state spaces.

Solving the planning and verification problems together at the same time is of interest in practical cases. For instance, in the management of natural renewable resources, one wants to find a policy that maximises the amount of resources that are harvested, while preserving the renewal so that the resources will still be available in the future. This is typically a problem where planning and verification are of equal importance. In another context, one may want to make sure that a communication system allows for the fast routing of messages, while preserving the privacy of users. This is again a problem of planning (having the fastest routes) and verification (checking that privacy is enforced).

In this paper, we focus on approximation methods based on randomized approximation algorithms. In this context, efficiency means two things: the first one is that the algorithm depends on a parameter ε which measures the quality of the approximation; the second one is the space complexity of the algorithm which is logarithmic in the size of some compact representation of the system. Indeed, the approximation method has only access to a probabilistic generator, or a generative model, of the Markov Decision Process. Given any state, the algorithm uses the probabilistic generator to draw samples for many state-action pairs, and uses these samples to compute a near-optimal action from the given state, which is then executed. Once an approximation of an optimal policy is obtained, the MDP restricted to this policy is a Markov Chain, on which one can efficiently compute a good approximation of the satisfaction probabilities of interesting properties, like reachability or safety.

We recall in section 2 the framework of MDPs with rewards, the planning problem for policies, the complexity of the policy iteration algorithm, a brief survey of probabilistic verification, the Chernoff-Hoeffding bounds for sampling and our method to approximate probabilistic verification of interesting quantitative properties over Markov Decision Processes. In section 3, we adapt Kearns's learning approach to obtain near-optimal policies for the planning problem. In section 4, we sketch an alternative algorithm using the Multiple Weight Update Method to accelerate the convergence toward a near-optimal policy. Approximate planning methods and randomized approximation methods for probabilistic verification are combined in section 5. Section 6 is dedicated to the related work.

2 Preliminaries

2.1 Markov Decision Processes

A Markov Decision Process (MDP) is a quadruple $\mathcal{M} = (S, A, P, R)$ where S is a finite (or countable) set of states, A is a finite set of actions, $P : S \times A \longrightarrow Distr(S)$, where $Distr(S)$ is the set of probability distributions over S, is the transition relation and $R : S \times A \longrightarrow \mathbb{R}^+$ is the reward function. Thus a MDP is defined by:

- for each state-action pair (s, a), a next-state distribution $P_{s,a}(.)$ that specifies the probability of transition from state s to another state when the action a is chosen,
- for each state-action pair (s, a), a real-valued reward $R_{s,a}$ for executing action a from state s.

The initial state of the system is chosen randomly according to an initial probability distribution on state space. The function $P_{s,a}$ can be extended to subsets $X \subseteq S$ by: $P_{s,a}(X) = \sum_{t\in X} P_{s,a}(t)$. Labeled Markov Processes, also called Probabilistic Labeled Transition Systems, are Markov Decision Processes without rewards.

A run on $\mathcal{M}$ is a finite or infinite sequence $r = (s_0, a_1, s_1, \ldots, s_{i-1}, a_i, s_i, \ldots)$ such that for any $i > 0$, $a_i \in A, s_i \in S$ and $P_{s_{i-1},a_i}(s_i) > 0$.

Given a run r, we can consider the associated execution path r_S which is the restriction of the run r to the sequence of states. Let r_A be the trace associated to r, i.e. the restriction of r to the sequence of actions. Given a run r and $n \in \mathbb{N}$, we write r_n for the sequence of the first $n-1$ states in r_S.

A stationary policy on $\mathcal{M}$ is a function $\pi : S \longrightarrow Distr(A)$ which resolves the non determinism of the system by choosing a distribution on the set of available actions for each state of the MDP. The notion of policy is closely related to the notions of scheduler [29], adversary [27] or strategy [5]. A policy π and an initial distribution $\alpha \in Distr(S)$ induce a probability distribution $\mathbb{P}_{\pi,\alpha}$ on the π-field $\mathcal{F}$ of the set of runs, generated by the cones $C_\sigma = \{r \mid r_{|\sigma|} = \sigma\}$, see [7, 29]. In the particular case of $|A| = 1$, *i.e.,* there is only one action possible, the MDP is in fact a Markov chain. When a stationary policy is fixed, the restriction of the MDP to this policy is also a Markov chain.

In order to overcome the complexity barrier and to preclude approaches that compute directly on a full representation of an MDP, we assume that an MDP is only given in the form of the ability to sample its behavior. We call this simulative form of the model a *probabilistic generator* . We note that the existence of a *succinct representation*, as in [23], clearly gives a probabilistic generator.

Definition 1. A probabilistic generator for an MDP $\mathcal{M}$ is a randomized algorithm that, on input of a state-action pair (s, a), outputs a state t, which is randomly drawn according to the next-state distribution $P_{s,a}(.)$, and the associated reward $R_{s,a}$.

This notion is well-known in the literature. For example, a probabilistic generator for an MDP is also called a *generative model* [21] or a *simulation model* in many simulation-based algorithms for MDPs [6]. Moreover, such succinct representations are used to represent MDPs via the input language of verification tools such as PRISM [17].

2.2 Planning problem in MDPs

In the following, we consider finite, or countable MDPs with the discounted total expected reward criterion, i.e. given a number $0 \leq \gamma < 1$ the value function $V^\pi : S \longrightarrow \mathbb{R}$ underlying a policy π is defined by:

$$V^\pi(s) = \mathbb{E}^\pi_s(\sum_{i=1}^{\infty} \gamma^{i-1} r_i)$$

where r_i is the reward received on the ith step of executing the policy π from state s, and the expectation is over the probability distribution $\mathbb{P}_{\pi,\alpha}$ and any randomization in π. Define the action-value function $Q^\pi : S \times A \longrightarrow \mathbb{R}$ underlying a policy π as:

$$Q^\pi(s, a) = R_{s,a} + \gamma \mathbb{E}_{s' \sim P_{s,a}(.)}(V^\pi(s'))$$

where the notation $s' \sim P_{s,a}(.)$ means that s' is drawn according to the distribution $P_{s,a}(.)$. The optimal value function V^*, optimal Q-function Q^* and optimal policy π^* are defined by: for all $s \in S$,

$$V^*(s) = sup_\pi V^\pi(s)$$
$$Q^*(s, a) = sup_\pi Q^\pi(s, a)$$
$$\pi^*(s) = argmax_a Q^*(s, a)$$

In the class of MDPs that we consider here, an optimal stationary policy always exists. The planning problem in MDPs is to compute such an optimal policy. This problem can be solved by linear programming or dynamic programming. In the latter approach, the algorithms that compute the optimal policy require storage for two arrays indexed by states for value function V and policy π. The algorithms have two kinds of steps:

(step 1)
$\pi(s) := argmax_a(R_{s,a} + \gamma \mathbb{E}_{s' \sim P_{s,a}(.)}(V(s')))$

(step 2)
$V(s) := R_{s,a} + \gamma \mathbb{E}_{s' \sim P_{s,\pi(s)}(.)}(V(s'))$

The two main types of algorithms are value iteration and policy iteration. In value iteration, substituting the calculation of $\pi(s)$ into the calculation of $V(s)$ gives the combined step:

$$V(s) := max_a(R_{s,a} + \gamma \mathbb{E}_{s' \sim P_{s,\pi(s)}(.)}(V(s')))$$

This update rule is iterated for all states s until it converges with the left-hand size approximatively equal to the right-hand size. However, the most common way of solving planning problem for MDPs in practice is Howard's policy iteration algorithm. In policy iteration, step 1 is performed once, and then step 2 is repeated until it converges. Then step 1 is again performed and so on.

These classical approaches use an explicit representation of a model of size N. They are thus intractable for large N, where even reading all the input can take $O(N^2)$ time and specifying a policy requires space on the order of N.

2.3 Policy Iteration is strongly polynomial

The problem of determining worst case complexity of Howard's algorithm was stated explicitly at least twenty-five years ago [19]. Recently, Ye presented a simple proof that Howard's algorithm terminates after at most $O(\frac{kn}{1-\gamma} log(\frac{n}{1-\gamma}))$, where n is the number of states, k is the number of actions and $0 < \gamma < 1$ is the discount factor (see [30]). In particular, when the discount factor is fixed, the number of iterations is $O(knlogn)$. Since each iteration only involves solving a system of linear equations, Ye's result established that Howard's algorithm is a strongly plolynomial time algorithm, when the discount factor is fixed.

More recently, Ye's analysis was improved and extended by T.D. hansen, P.B. Miltersen and U. Zwick to two-player turn-based stochastic games (see [13]). They showed that Howard's algorithm terminates after at most $O(\frac{k}{1-\gamma} log(\frac{n}{1-\gamma}))$ iterations, improving Ye's bound by a factor of n. The way to obtain this result is a well-known relationship between Howard's policy iteration algorithm and Bellmann's value iteration algorithm.

Ye's and Zwick's results, combined with the recent results of Friedmann [10] and Fearnley [9] give a complete characterization of the complexity of the policy iteration algorithm. The policy iteration algorithm is strongly polynomial for a fixed discount factor, but exponential for non discounted MDPs.

2.4 Probabilistic Verification

The first application of verification methods to probabilistic systems consists in checking if temporal properties are satisfied with probability 1 by systems modeled either as finite probabilistic transition systems, *i.e.,* discrete-time Markov chains, or as concurrent probabilistic transition systems, *i.e.,* MDPs. This problem was called qualitative verification. In [29], Vardi presented the first method to verify if a linear time temporal property is satisfied by almost all computations of a concurrent probabilistic system. This automata-theoretic method is expensive, since its complexity is doubly exponential in the size of the formula.

The complexity of this work was later addressed by Courcoubetis and Yannakakis [7]. A new model checking method for probabilistic systems was introduced, the complexity of which was proved polynomial in the size of the system and exponential in the size of the formula. For concurrent probabilistic systems they presented an automata-theoretic approach which improved on the Vardi's method by a single exponential in the size of the formula.

Courcoubetis and Yannakakis's method [7] also solves the quantitative verification problem, *i.e.,* to compute the probability that a probabilistic system, modeled as a Markov chain, satisfies some given linear time temporal formula. The algorithm transforms step by step the Markov chain and the formula, eliminating one by one the temporal connectives, while preserving the satisfaction probability of the formula. The elimination of temporal connectives is performed by solving a linear system of equations of the size of the Markov chain.

A model checking method for properties expressed in a branching time temporal logic extended by a probabilistic operator against concurrent probabilistic systems is given by Bianco and de Alfaro [5]. Probabilities are computed by solving an optimisation problem over a system of linear inequalities, the size of which is the model size.

In order to illustrate space complexity problems, we mention the main model checking tool for the verification of quantitative properties. PRISM [8,22] allows to check linear and branching time temporal formulas extended with a probabilistic operator on probabilistic or concurrent probabilistic systems. PRISM also supports costs and rewards. However, in many applications MDPs are very large and the computation is intractable.

For theoretical and practical reasons, it is natural to ask the question: *can probabilistic verification be efficiently approximated?* Approximate probabilistic verification has been investigated for Markov chains in [15,23]. In [17], the authors of PRISM integrated a version of Approximate Probabilistic Model Checking

(APMC) to obtain a simulator that can be used in case of very large MDPs. In section 3, we describe some approximation method for planning and verification of interesting quantitative properties expressed in linear time temporal logic over large or infinite MDPs.

2.5 The Chernoff-Hoeffding bounds for sampling

In section 3, we will use Chernoff-Hoeffding bounds [18] on the tail of the distribution of a sum of independent random variables. The main interest of Chernoff-Hoeffding bounds is to allow sampling procedures with polynomial size in order to estimate probabilities or expectations.

Lemma 1. *[18] Let $X_1, ..., X_N$ be N independent random variables which take value 1 with probability p and 0 with probability $(1-p)$, and $Y = \sum_{i=1}^{N} X_i/N$. Then the Chernoff-Hoeffding bound gives:*

$$Prob\big(|Y-p|\big) \geq 1 - 2e^{-2N\cdot\varepsilon^2}.$$

The approximation will be correct, *i.e.,* $|Y - p| < \varepsilon$, with confidence $(1-\delta)$, after a sampling of polynomial size N in $\frac{1}{\epsilon}, \log\frac{1}{\delta}$, *i.e.,*

$$N = \ln(\frac{2}{\delta})/2\varepsilon^2.$$

2.6 Approximate Probabilistic Verification

For many linear time properties, as reachability, satisfaction by an execution path of finite length implies satisfaction by any extension of this path. Such properties are called monotone. Another important class of properties, namely safety properties, can be expressed as negation of monotone properties. We can reduce the computation of satisfaction probability of a safety property to the same problem for its negation, that is a monotone property. In [23], it is demonstrated that the satisfaction probability of monotone or anti-monotone linear time properties can be approximated with a randomized approximation scheme. This means that, given a Discrete Time or a Continuous Time Markov Chain (DTMC or CTMC) $\mathcal{M}$ and a monotone property ψ, it is possible to approximate $Prob[\psi]$, the probability measure of the set of execution paths satisfying the property ψ by a fixed point algorithm obtained by iterating a randomized approximation scheme for $Prob_k[\psi]$, the probability measure associated to the probabilistic space of execution paths of finite length k. For that purpose, the authors of [23] adapt the notion of randomized approximation scheme for counting problems, which is due to Karp and Luby [20], to the problem of computing probabilities. A probability problem has the goal, given a probabilistic generator of a probabilistic system and a linear time property x, to compute the probability measure $\mu(x)$ of the measurable set of execution paths satisfying this property.

Definition 2. [23] A randomized approximation scheme for a probability problem is a randomized algorithm $\mathcal{A}$ that takes an input x, two real numbers $\varepsilon, \delta > 0$ and produces a value $A(x, \varepsilon, \delta)$ such that:

$$Pr\Big(|A(x,\varepsilon,\delta) - \mu(x)| < \varepsilon\Big) \;\geq\; 1-\delta.$$

If the running time of $\mathcal{A}$ is polynomial in $|x|$, $\frac{1}{\varepsilon}$ and $\log(\frac{1}{\delta})$, $\mathcal{A}$ is said to be fully polynomial.

In [23], a probabilistic generator is used to generate random paths and to compute a random variable Y which approximates $p = Prob_k[\psi]$. The approximation will be correct, i.e $|Y - p| < \varepsilon$ (additive error), with confidence $(1-\delta)$, after a polynomial number of samples in $\frac{1}{\epsilon}, \log\frac{1}{\delta}$.

The following random sampling algorithm, called $\mathcal{GAA}$ for Generic Approximation Algorithm, uses a probabilistic generator G to compute a good approximation of $Prob_k[\psi]$.

```
$\mathcal{GAA}$
**Input:** $G, k, \psi, \varepsilon, \delta$
**Output:** $\varepsilon$-approximation of $Prob_k[\psi]$
$N := \ln(\frac{2}{\delta})/2\varepsilon^2$
$A := 0$
For $i = 1$ to $N$ do
1. Generate a random path $\sigma$ of length $k$
2. If $\psi$ is true on $\sigma$ then $A := A + 1$
Return $Y = A/N$
```

The following result is obtained by using a Chernoff-Hoeffding bound [18] on the tail of the distribution of a sum of independent random variables.

Theorem 1. *[23] The generic approximation algorithm $\mathcal{GAA}$ is a fully polynomial randomized approximation scheme for computing $p = Prob_k[\psi]$ whenever ψ is a monotone or anti-monotone linear time property and $p \in]0,1[$.*

As a corollary, the fixed point algorithm defined by iterating the approximation algorithm $\mathcal{GAA}$ is a randomized approximation scheme, whose space complexity is logspace, to compute the probability $Prob[\psi]$ for monotone or anti-monotone linear time properties.

The generic approximation algorithm $\mathcal{GAA}$ was implemented in several verification tools, first in APMC (Approximate Probabilistic Model Checker, see [16]), then in PRISM (see [17]). Since the algorithm processes random paths independently, the algorithm can benefit from parallel and distributed architectures [11,12].

3 Approximate Planning using sampling methods

3.1 Planning problem and Learning in MDPs

The learning phase of our first planning and verification method is an adaptation of Kearns's learning algorithm [21]. The running time and the space complexity of the learning phase are independent of the number of states of $\mathcal{M}$. Given as input a state s, the algorithm presented in [21] uses a probabilistic generator to find a near-optimal action to perform from state s. The basic idea of the method is to sample the probabilistic generator from states in the neighborhood of s.

In this paper, our goal is to build a Markov chain $\mathcal{M}^*$ obtained by restricting the MDP $\mathcal{M}$ to a near-optimal policy π^*. Following the Kearns's idea, we construct a small MDP $\mathcal{M}'$ of depth H'. For any state u of $\mathcal{M}'$ at depth at most $H = H'/2$, we ensure that the optimal action in $\mathcal{M}'$ from u is a near-optimal action from u in $\mathcal{M}$. $\mathcal{M}'$ is built as a sparse look-ahead directed tree in which the start state is s, and in which taking an action from a node in the tree causes a transition to a sample of C random children of that node with the corresponding action label. In order to build $\mathcal{M}^*$, $\mathcal{M}'$ is extended with some additional information that are given below. The main property of the tree is that the size of $\mathcal{M}'$ can be independent of the number of states in $\mathcal{M}$. This is obtained by establishing bounds on the required depth H of the tree and the required degree C of each node in the tree.

Rather than estimating the optimal value function V^*, the algorithm estimates, for a value of H to be specified later, the H-step expected discounted reward $V_H^*(s)$, given by:

$$V_H^*(s) = \mathbb{E}_s^{\pi^*}(\sum_{i=1}^{H} \gamma^{i-1} r_i)$$

where r_i is the reward received on the ith step upon executing the optimal policy π^* from state s. Moreover, the h-step expected discounted reward $V_h^*(s)$, for $h \geq 1$, is recursively given by:

$$V_h^*(s) = R_{s,a^*} + \gamma \mathbb{E}_{s' \sim P_{s,a^*}(.)}(V_{h-1}^*(s'))$$

$$V_h^*(s) \approx max_a\{R_{s,a} + \gamma \mathbb{E}_{s' \sim P_{s,a^*}(.)}(V_{h-1}^*(s'))\}$$

where a^* is the action taken by the optimal policy from state s and $V_0^*(s) = 0$.

The algorithm will approximate the expectation in this last equation by a sample of C random next states from the probabilistic generator. We modify the recursive procedure in Kearns's learning algorithm to obtain an estimation $\hat{V}_h^*(s)$ of $V_h^*(s)$. More precisely, we construct an intermediary structure $\widehat{\mathcal{M}}$ according to the following:

– storing for each node s in the tree representing the MDP $\mathcal{M}'$, for each action $a \in A$, the set $S_{s,a}$ of the C successors of (s) according to the next-state distribution $P_{s,a}(.)$. Note that the first time a state s is encountered in this process, we pick the C successors belonging to each action. Once this is done, all sets $S_{s,a}$ (for this state s and for all $a \in A$) are stored and reused when s is considered again.
– tagging the action a corresponding to the maximum value in the previous equation. We denote this action by a^*.

This extension of $\mathcal{M}'$ is from now denoted as $\widehat{\mathcal{M}}$ and is depicted in figure 1. s denotes the initial node and a^* is written as a superscript of actions of the near-optimal policy $\widehat{\pi}^\star$. It is worth noting that the near-optimal policy $\widehat{\pi}^\star$ is Markovian.

These additional information allow to build the Markov chain $\mathcal{M}^*$ by deleting nodes and transitions corresponding to non tagged actions and also by pruning the branches of our intermediary structure $\widehat{\mathcal{M}}$ to a certain depth H.

Bounding the required depth of the tree is the easy part. Let the so-called ε-horizon time be $H' = 2H$ where

$$H = log_\gamma(\lambda/V_{max}),$$

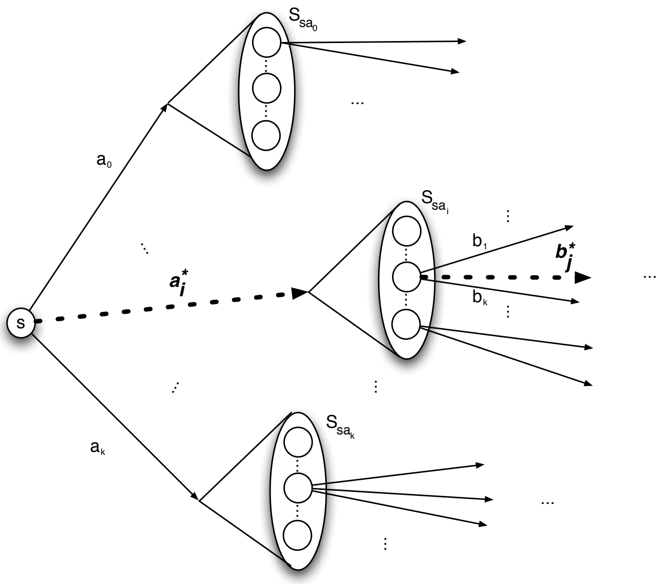

Fig. 1: Structure of $\widehat{\mathcal{M}}$

with $\lambda = \varepsilon(1-\gamma)^2/4$ and $V_{max} = R_{max}/(1-\gamma)$. H is the depth used in Kearns's construction. Our intermediary structure $\widehat{\mathcal{M}}$ has depth H', so for every state of the chain $\mathcal{M}^*$, the H-step expected discounted reward $V_H^*(s)$ is an ε-approximation of the optimal value.

By using a Chernoff-Hoeffding bound, we can obtain that, at each step, the probability of a single bad estimate for $Q^*(s,a)$ is $e^{-\lambda^2 C/V_{max}^2}$. The probability of a bad estimate increases by a factor of kC at each step, where k is the number of actions. Therefore the probability of some bad estimate after H steps is bounded by $(kC)^H e^{-\lambda^2 C/V_{max}^2}$. We can choose C so that this last quantity is bounded by λ/R_{max}. The main property of the degree C of each node in the tree is that it can be chosen independent of the number of states in $\mathcal{M}$. The key argument is that even though small samples may give very poor approximations to the next-state distribution at each state in the tree, they will, nevertheless, give good estimates of the expectation terms in the last equation.

However, as we shall explain in the section 5, even though the running time of the Kearns's learning algorithm does not depend on the size of the MDP, it still runs in time exponential in the ε-horizon time, and therefore exponential in $1/(1-\gamma)$ whenever γ is very close to 1. In this case, the use of another sampling technique such as policy rollout can be envisioned.

3.2 Rollout-based Monte-Carlo Planning

In order to improve the performance of the Approximate Planning algorithm, one can apply a particular bandit algorithm, named UCB (Upper Confidence Bounds, see [3]), for rollout-based Monte-Carlo planning. As opposed to the previous planning algorithm, a rollout-based algorithm builds its lookahead tree by repeatedly sampling episodes from the initial state. An episode is a sequence of state-action-reward triplets that are obtained using the generative domain. The reason to consider rollout-based algorithms is that they allow to keep track of estimates of the actions' values at the sampled states encountered in earlier episodes.

The algorithm iteratively generates episodes and returns the action with the highest average long-term reward. Episodes are generated by a search function that selects and effectuates actions recursively until the reach of a terminal state, or episodes can be cut at a certain depth. The search function computes the total reward corresponding to the episode and the reward value is used to adjust the estimated value for the given state-action pair at the given depth, completed by increasing the counter that stores the number of visits of this state-action pair.

The main feature of this algorithm is the introduction of a bandit algorithm for the implementation of the selective sampling of actions.

A bandit problem with K actions, or arms, is defined by a sequence of rewards $X_{it}, i = 1, \ldots, K, t \geq 1$ where each i is the index of an action. Successive choices of action i yield the rewards $X_{i1}, R_{i2}, \ldots$. For simplicity, one can assume that all rewards X_{it} lie in the interval $[0, 1]$. An allocation policy is a mapping that selects the next action to be chosen based on the sequence of past selections and associated obtained rewards. The expected regret of an allocation policy after n choices is defined by:

$$reg_n = max_i(\sum_{t=1}^{n} X_{it}) - (\sum_{j=1}^{K} \sum_{t=1}^{T_j(n)} X_{it})$$

where $I_t \in \{1, \ldots, K\}$ is the index of the action selected at time t by the policy, and where $T_i(t)$ is the number of times action i was played up to time t. Thus, the regret is the loss caused by the policy that does not always choose the best action. The algorithm UCB keeps track of the average rewards $\bar{X}_{i,T_i(t-1)}$ for all the actions and thus can choose the action with the best upper confidence bound:

$$I_t = argmax_{i \in \{1,\ldots,K\}}(\bar{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)})$$

where $c_{t,s}$ is a bias sequence chosen to be $c_{t,s} = \sqrt{2lnt/s}$.

The bias sequence is such that:

$$Prob[\bar{X}_{is} \geq \mu_i + c_{t,s}] \leq t^{-4},$$

$$Prob[\bar{X}_{is} \leq \mu_i + c_{t,s}] \leq t^{-4}$$

This comes directly from the Chernoff-Hoeffding's inequality. The fundamental consequence of this result is that the regret after n successive choices is logarithmic with respect to n.

4 Approximate policies using the Multiple Weights Update Method

The Multiple Weights Update Method (MWUM) [2] is used in various fields to speed up the convergence of optimization algorithms. In the context of approximate planning for large MDPs, the idea is to maintain a distribution over the set of stationary deterministic policies and to use the multiplicative update rule to iteratively change the weights in order to obtain a near-optimal policy. Initially, the algorithm select uniformly at random amongst the policies. As time goes on, some policies are seen as giving better total rewards than others, and the algorithm increases their weights proportionally. The multiplicative update rule is thus the way to improve drastically the distribution.

We note $M(\pi_i, j)$ the outcome, which is the total reward, that a stationary deterministic policy $\pi_i (1 \leq i \leq m)$ gives for the event j corresponding to the successive choices state/action made by the policy π_i to resolve the non determinism up to the finite horizon H. For each step t of the algorithm, we consider the distribution $D_t = \{p_1^t, p_2^t, \ldots, p_m^t\}$ on the set of policies $\pi_i (1 \leq i \leq m)$. The probabilities $p_i^t (1 \leq i \leq m)$ are computed at each step according to the multiplicative update rule. At each step t, we can consider the randomized policy defined by applying the policies $\pi_i (1 \leq i \leq m)$ according to the distribution D_t.

```
$\mathcal{MWUA}$
**(Multiple Weights Update Algorithm)**
**Input:** $\varepsilon$, T, MDP $\mathcal{M}$
**Output:** weights $w_i^T()$
$w_i^1 := 1$ for all $1 \leq i \leq m$
For $t = 1$ to $T - 1$ do
    For $1 \leq i \leq m$
        1. $p_i^t := \frac{w_i^t}{\sum_{i=1}^{m} w_i^t}$
        2. $w_i^{t+1} := w_i^t (1 - \varepsilon)^{M(\pi_i, j_t)/\rho}$
    EndFor
EndFor
Return $w_i^T (1 \leq i \leq m)$
```

where $0 < \varepsilon \leq 1/2$ is an approximation parameter, j_t is the event at step t and ρ is the maximum total reward.

The expected reward for event j_t is

$$\sum_{i=1}^{m} p_i^t M(\pi_i, j_t)$$

which we denote by $M(D_t, j_t)$. By linearity of expectations, the expected total reward is $\sum_{t=1}^{T} M(D_t, j_t)$.

Theorem 2. *Let $0 < \varepsilon \leq 1/2$. After T steps, for any stationary deterministic policy $\pi_i (1 \leq i \leq m)$, we have:*

$$\sum_{t=1}^{T} M(D_t, j_t) \geq (1 - \varepsilon) \sum_{t=1}^{T} M(\pi_i, j_t)) - \frac{\rho lnm}{\varepsilon}$$

Let π^* be an optimal stationary deterministic policy. As a consequence of the previous result, we obtain:

$$\sum_{t=1}^{T} M(\pi^*, j_t) \geq \sum_{t=1}^{T} M(D_t, j_t)$$

and

$$\sum_{t=1}^{T} M(D_t, j_t) \geq (1-\varepsilon)\sum_{t=1}^{T} M(\pi^*, j_t) - \frac{\rho lnm}{\varepsilon}.$$

Corollary 1. *Let $0 < \delta$. After $T = \frac{4\rho^2 lnm}{\delta^2}$ steps, the average total reward is a good additive approximation of the total reward given by an optimal stationary deterministic policy:*

$$\frac{1}{T}\sum_{t=1}^{T} M(\pi^*, j_t) \geq \frac{1}{T}\sum_{t=1}^{T} M(D_t, j_t)$$

and

$$\frac{1}{T}\sum_{t=1}^{T} M(D_t, j_t) \geq \frac{1}{T}\sum_{t=1}^{T} M(\pi^*, j_t) - \delta.$$

5 Approximate Planning and Probabilistic Verification

Taking into account that an approach with an explicit representation of an MDP is clearly infeasible for very large state spaces, we propose an efficient approximation method for planning and verification when a MDP is given in the form of a probabilistic generator, which has the ability to simulate its behavior.

The approximation method is working in two phases: the planning phase and the verification phase. The first one is a randomized algorithm to compute a near-optimal policy for the planning problem in discounted MDPs. The second one is a *randomized approximation scheme* to approximate satisfaction probabilities of linear time properties over the Markov chain when restricting the MDP to the near-optimal policy. For simplicity, we described the two phases separately, but they are combined in the resulting algorithm.

5.1 Approximate Planning by learning and Probabilistic Verification

Our final goal is to approximate the satisfaction probability of a monotone or anti-monotone linear time formula on a MDP under a near-optimal policy $\widehat{\pi}^*$. This probability is denoted by $Prob[\psi|\widehat{\pi}^\star]$.

The randomized algorithm that computes a near-optimal policy for the planning problem in discounted MDPs is written in figure 2.

We remind the reader that the tagged actions are those of the near-optimal policy $\widehat{\pi}^*$. For the sake of clarity, we illustrate in figure 3 the step 3 of $\mathcal{APVA}$. Each action of $\widehat{\mathcal{M}}$ is considered. If the action is one of the actions of $\widehat{\pi}^*$ we keep the action for building $\mathcal{M}^*$, otherwise we erase the transition labelled by the action together with the whole sub-tree induced by this transition in $\widehat{\mathcal{M}}$. It should be noted that each state of $\mathcal{M}^*$ has only one leaving action remaining after the trimming stage. It means that all non-deterministic steps have disappeared and that $\mathcal{M}^*$ is a Markov chain (remind that the near-optimal policy is Markovian).

Recall from [21] that the horizon H and width C parameters are given by:

$$H = \lceil log_\gamma(\lambda/V_{max})\rceil$$

$$\text{where } \lambda = \varepsilon(1-\gamma)^2/4$$

$$\text{and } V_{max} = R_{max}/(1-\gamma)$$

$$C = O(\frac{V_{max}^2}{\lambda^2}(2H \cdot logH + log(R_{max}/\lambda))$$

We remark that the size C^H of the set of execution paths of length H in the Markov chain $\mathcal{M}^*$ is much more large than the sample size in the generic approximation algorithm $\mathcal{GAA}$. The following result is obtained as a consequence of the two previous subsections.

Theorem 3. *The approximate verification algorithm $\mathcal{APVA}$ is a randomized algorithm that, given access to a probabilistic generator for any k-action MDP $\mathcal{M}$ with discount factor γ, takes as input a state s, a linear time formula ψ and two real parameters ε, δ and satisfies the following conditions:*

- *it is a randomized approximation scheme for computing $p = Prob[\psi|\widehat{\pi}^\star]$ whenever ψ is a monotone or anti-monotone linear time property and $p \in]0,1[$,*
- *the value function of the near-optimal stochastic policy $\widehat{\pi}^\star$ is an ε-approximation of $V^\star$, i.e. $|\widehat{V}(s) - V^\star(s)| \leq \varepsilon$, with probability greater than, or equal to, $1-\lambda/R_{max}$,*
- *the running time is $O((kC)^H)$ and the space complexity is $O(C^H)$.*

$\mathcal{APVA}$ provides a different complexity tradeoff than classical algorithms for planning problem and probabilistic verification in MDPs. Usually, classical algorithms as value iteration for

Approximate planning and verification algorithm $\mathcal{APVA}$
Input: $G, \gamma, s, \psi, \varepsilon, \delta$
Output: ε-approximation of $Prob[\psi|\widehat{\pi}^{\star}]$
1. Compute horizon H and width C as functions of ε, γ and of the maximum reward R_{max}.
2. Construct $\widehat{\mathcal{M}}$ from G
3. Trim $\widehat{\mathcal{M}}$ to obtain $\mathcal{M}^*$ by deleting branches corresponding to non tagged actions.
4. Apply the generic approximation algorithm $\mathcal{GAA}$ to $(\mathcal{M}^*, H, \psi, \varepsilon, \delta)$

Fig. 2: $\mathcal{APVA}$

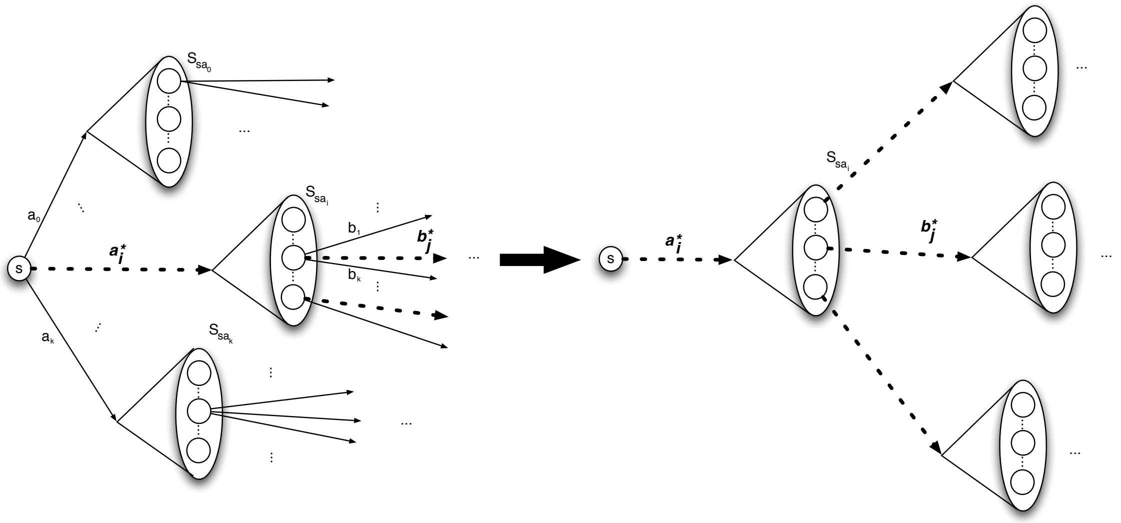

Fig. 3: Step 3 of the $\mathcal{APVA}$ algorithm

planning or solving linear systems for probabilistic verification have a polynomial time complexity with respect to the state space size and are unusable for very large models. In contrast to this complexity, the complexity of the $\mathcal{APVA}$ algorithm does not depend of the state space size and is polynomial with respect to $1/\varepsilon$ and $log(1/\delta)$, where ε is the error parameter and $(1-\delta)$ the confidence parameter.

However, for practical issues, even though the running time of the algorithm $\mathcal{APVA}$ does not depend on the size of the MDP, it still runs in time exponential in the ε-horizon time, and therefore exponential in $1/(1-\gamma)$ whenever γ is very close to 1.

5.2 Approximate Planning by MWUM and Probabilistic Verification

In this section, we present a possible use of MWUM for the joint planning and verification problem. We call our method $\mathcal{MWPV}$ for Multiple Weights for Probabilistic Verification. This is a first attempt to use multiplicative weights in this context, and our approach is clearly not optimal. It can however lead to interesting practical results. Here again, the method proceeds in two steps. First, we find, using the multiple weights update algorithm, a distribution over deterministic policies that defines a good policy $\pi^{\star}$ with respect to maximizing the total reward one can obtain over traces of a given system. Rewards are either discounted or the horizon is finite, thus we only have to consider the total reward over finite traces of a given length.

We recall that a stationary deterministic policy is a restriction of policies. In a deterministic policy π, each state is given a unique action that resolves the non-determinism. This means that, for a stationary deterministic policy π and $\forall s \in S$, $\pi(s) = Distr(next(s))$, where $next(s)$ is the set of states reachable from s.

Once we have obtained $\pi^\star$, the second step of the method uses it in order to compute $Prob[\psi|\pi^\star]$, where ψ is a LTL formula. For this step, we use the generic approximation algorithm over paths generated using $\pi^\star$ and random choices over reachable states.

In the following we present in more details the method, and then discuss its practical aspects.

5.2.1 Multiple Weights for Probabilistic Verification

The first step of our method $\mathcal{MWPV}$ has the goal of computing $\pi^\star$. The algorithm by itself is depicted in figure 4. Amongst the input of the algorithm, ε and δ are approximation parameters, and $\mathcal{M}$ is the MDP. m denotes the number of deterministic policies over this MDP. We assume that we have a data structure that stores all the deterministic policies, *i.e.,* we know, for each deterministic policy π_i and state s_j, the transition probabilities for states reachable from s_j under the action chosen by π_i. We also make the hypothesis that the horizon is known, we denote it by H. This is realistic when rewards are discounted, or when the system has a natural bound on its execution.

Note that we have replaced ρ (the maximum reward) by the quantity max_total. Indeed, since we access $\mathcal{M}$ through its succinct representation, we can only give an upper bound to the maximum total reward (basically it is $H \cdot R_{max}$). Moreover, the algorithm uses a quantity that we note $total\_reward(i,t)$. The computation of this quantity is the keypoint of our algorithm.

In order to update the weights, we need to know the outcome of the choices of a given policy. The method proceeds using a fictitious play approach. For any given t, suppose that we are looking at policy π_i. The total reward is initialized to $total\_reward(i,t) = 0$. Starting from the initial state of $\mathcal{M}$, the method is doing a traversal of $\mathcal{M}$ until it reaches the horizon H. At state s_j, it chooses the next state at random according to $\pi_i(s_j)$ and updates $total\_reward(i,t)$ by adding to it the local reward obtained at state s_j. At the end of the traversal, the outcome of the policy π_i is $total\_reward(i,t)$.

Using this algorithm, we obtain a probability distribution over deterministic policies that defines $\pi^\star$. More precisely, we obtain a probability distribution over probabilistic transition matrices over S. We can combine these information into a unique probabilistic transition matrix over S. This matrix represents the DTMC that corresponds to the use of the policy $\pi^\star$ on $\mathcal{M}$. Then, we can apply the generic approximation algorithm $\mathcal{GAA}$ to this DTMC in order to compute $Prob[\psi|\pi^\star]$ for any LTL formula ψ.

5.2.2 Practical aspects

The main issue concerning our method $\mathcal{MWPV}$ is that its time complexity is highly dependent on m, the number of deterministic policies in the MDP $\mathcal{M}$. Moreover, the space complexity suffers from the same problem since it is needed to store the probabilities for any of the deterministic policies.

While this is a problem from the theoretical point of view, this is unlikely to be an issue while analyzing real life systems. Indeed, typical industrial systems have only very few non deterministic steps, making the number of deterministic policies highly tractable.

Moreover, for MDPs that can be represented using high level languages such as Reactive Modules [1] or PRISM input language, symmetry and/or code factoring is very important. This means that most of the time it will be possible to encode $\pi^\star$ on the fly directly in the succinct representation.

Finally, the main interest of this method is that it is always tractable to use it. Indeed, once you have chosen the value of T, it is always possible to compute a corresponding $\pi^\star$. This means that if one knows how much computation power one has, it is possible to use the method, with an approximate result that will be the best that can be obtained with this actual computation power.

6 Related work

Several methods exist for the planning problem in MDPs, but most are less efficient than sampling based methods. For years, these sampling-

```
$\mathcal{MWPV}$ – **step 1**

**Input:** $\mathcal{M}, \varepsilon, \delta$
**Output:** probabilities $p_i^T()$

For all $1 \leq i \leq m$, $w_i^1 := 1$
$t := 1$
$T := \frac{4(max\_total)^2 ln m}{\delta^2}$

While $t \leq T$ do
  For $1 \leq i \leq m$ do
        $p_i^t := \frac{w_i^t}{\sum_{i=1}^m w_i^t}$
        $w_i^{t+1} := w_i^t (1-\varepsilon)^{\frac{total\_reward(i,t)}{max\_total}}$
  endFor
t := t+1
endWhile

For all $1 \leq i \leq m$, Return $p_i^T (1 \leq i \leq m)$
```

Fig. 4: $\mathcal{MWPV}$

based methods had the drawback of not having any known proof of their efficiency. This is no longer the case. Roughly, there are two types of sampling methods to approximate the planning problem in very large MDPs.

The first one is the sparse sampling technique initiated by Kearns *et al.* (see [21]) and the planning phase of our first approximation method uses an extension of Kearns's technique. The main advantage of sparse sampling is to provide strong theoretical guarantees for near optimality and complexity independent of state-space size. The main drawback is complexity exponential in $1/(1-\gamma)$ when the discount rate γ is close to 1.

The second type of sampling methods is policy rollout, an online version of policy iteration (see [28,4]). The amount of sampling required to guarantee policy improvement is independent of state-space size. Policy rollout is an easy way to improve policy quality from a simple initial policy, but there is no guarantee for near optimality.

Concerning the use of the multiple weights update method, it has been used previously for machine learning and game theory. But to the best of our knowledge, this is the first time it is used for both planning and verifying a system defined as a MDP.

Recently, a few papers adress the problem of the approximate verification of large MDPs. This paper is an extension of [24], which adresses both the planning and verification problems of MDPs. Some other research groups obtained results about the verification problem (they do not consider the planning problem). For instance, the authors of [14] present an algorithm that resolves nondeterminism probabilistically using sampling and Reinforcement Learning. Their algorithm allows to check the satisfaction of Bounded Linear Temporal Logic (BLTL) properties. Another interesting work has been presented by Legay and Sedwards [25]. They designed a lightweight Monte Carlo algorithm for the statistical model checking of models that contain non determinism. One of the advantages of their method is that it can be efficiently parallelised.

7 Discussion and conclusion

We presented two approaches for the design of an efficient randomized approximation method to combine planning and verification for very large MDPs.

In this context, we have used sparse sampling and randomized approximation schemes to compute a near-optimal policy for MDPs and to approximate the satisfaction probabilities of interesting properties over the Markov chain obtained by restricting the MDP to the near-optimal policy. To the best of our knowledge, it is the first time that the combination of Kearns's learning method and randomized approximation scheme is used for planning and verification in very large MDPs.

We also presented a prospective algorithm based on the Multiplicative Weights Update Method. This approach has a very high potential that is still to be fulfilled.

These two methods have their advantages and drawbacks. But there is a main difference between them. The first method, based on sparse sampling, uses the quality of the approximation as control criterion, while the second method ($\mathcal{MWPV}$) considers the speed of convergence as control criterion. This makes the second method more interesting in a real life context, where the computation power available for a verification task is known, rather than the quality one wants to achieve.

References

1. Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
2. Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.
3. Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.
4. Dimitri P Bertsekas and David A Castanon. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, 1999.
5. Andrea Bianco and Luca De Alfaro. Model checking of probabalistic and nondeterministic systems. In *Proc. 15th conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 1995.
6. Hyeong Soo Chang, Michael C Fu, Jiaqiao Hu, and Steven I Marcus. A survey of some simulation-based algorithms for markov decision processes. *Communications in information and systems*, 7(1):59–92, 2007.
7. Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM (JACM)*, 42(4):857–907, 1995.
8. Luca De Alfaro, Marta Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the kronecker representation. In *Proc. of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, pages 395–410, 2000.
9. John Fearnley. Exponential lower bounds for policy iteration. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 551–562. Springer, 2010.
10. Oliver Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 145–156. IEEE Computer Society, 2009.
11. Guillaume Guirado, Thomas Hérault, Richard Lassaigne, and Sylvain Peyronnet. Distribution, approximation and probabilistic model checking. *Electr. Notes Theor. Comput. Sci. - Proc. of Parallel and Distributed Model Checking (PDMC)*, 135(2):19–30, 2006.
12. Khaled Hamidouche, Alexandre Borghi, Pierre Esterie, Joel Falcou, and Sylvain Peyronnet. Three high performance architectures in the parallel approximate probabilistic model checking boat. In *Proc. of Parallel and Distributed Model Checking (PDMC)*, 2010.
13. Thomas Dueholm Hansen, Peter Bro Miltersen, and Uri Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 253–263. Tsinghua University Press, 2011.
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 (QEST), 2012 Ninth International Conference on*, pages 84–93, Sept 2012.
15. Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. In *Proceedings of the 5th Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 2937 of *Lecture Notes in Computer Science*, pages 73–84. Springer, 2004.
16. Thomas Herault, Richard Lassaigne, and Sylvain Peyronnet. APMC 3.0: Approximate verification of discrete and continuous time markov chains. In *Third International Conference on the Quantitative Evaluation of Sys-*

tems (QEST), pages 129–130. IEEE Computer Society, 2006.

17. Andrew Hinton, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 3920 of *Lecture Notes in Computer Science*, pages 441–444. Springer, 2006.
18. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
19. Ronald A Howard. *Dynamic programming and Markov process.* MIT Press, 1960.
20. Richard M Karp and Michael Luby. Monte-carlo algorithms for enumeration and reliability problems. In *Proc. of the 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 56–64. IEEE, 1983.
21. 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.
22. Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 2.0: A tool for probabilistic model checking. In *Proc. 1st International Conference on Quantitative Evaluation of Systems (QEST)*, pages 322–323. IEEE Computer Society Press, 2004.
23. Richard Lassaigne and Sylvain Peyronnet. Probabilistic verification and approximation. *Annals of Pure and Applied Logic*, 152(1-3):122 – 131, 2008.
24. Richard Lassaigne and Sylvain Peyronnet. Approximate planning and verification for large markov decision processes. In *Proceedings of the ACM Symposium on Applied Computing, SAC 2012*, pages 1314–1319. ACM, 2012.
25. Axel Legay and Sean Sedwards. Lightweight monte carlo algorithm for markov decision processes. *arXiv preprint arXiv:1310.3609*, 2013.
26. Martin L. Putterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley and Sons, 1994.
27. Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
28. Gerald Tesauro and Gregory R Galperin. Online policy improvement using monte-carlo search. *Advances in Neural Information Processing Systems*, pages 1068–1074, 1997.
29. Moshe Y Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. of the 26th Foundations of Computer Science (FOCS)*, pages 327–338, 1984.
30. Yinyu Ye. A new complexity result on solving the markov decision problem. *Mathematics of Operations Research*, 30(3):733–749, 2005.