Approximate Planning and Verification for Large Markov Decision Processes

Richard Lassaigne
Equipe de Logique Mathématique
Université Paris 7
Paris, F-75205, France

Sylvain Peyronnet
Univ Paris-Sud; LRI; UMR 8623
Orsay, F-91405, France

ABSTRACT

We study the planning and verification problems for very large or infinite probabilistic systems, like 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 of 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. The complexity of the approximation method is independent of the size of the state space and uses only a probabilistic generator of the MDP.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Model checking, Statistical methods*

General Terms

Verification

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 are used to model control problems for communications systems and industrial software-based control systems. In such a model, 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 types of 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. On the other hand, discounted MDPs are common in planning and unusual in verification.

In this paper, we study these two problems from a complexity point of view. For very large or infinite 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 restriction to the near-optimal policy. Classical methods for planning, such as value iteration and policy iteration [18], are intractable in the case of large state spaces.

In this paper, we focus on approximation methods based on sampling algorithms. In this context, efficiency means two things: the first one is that the algorithm takes also as input 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, a brief survey of probabilistic verification and the Chernoff-Hoeffding bounds for sampling. In section 3, we combine a learning approach to the planning problem and randomized approximation methods to check interesting quantitative properties over Markov Decision Processes. Section 4 is dedicated to the related work.

SAC'12 March 25-29, 2012, Riva del Garda, Italy.

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
$r = (s_0, a_1, s_1, \dots, s_{i-1}, a_i, s_i, \dots)$ 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 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 [21], adversary [19] or strategy [2]. 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 [4, 21]. 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 [17], 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* [15] or a *simulation model* in many simulation-based algorithms for MDPs [3] .

2.2 Planning problem in MDPs

In the following, we consider discounted MDPs, i.e. given a number $0 \leq \gamma < 1$ the value function V^π is defined for any policy π:

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

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. The Q-function is also defined for a given 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 $s \in S$,

$$V^*(s) = sup_\pi V^\pi(s)$$

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

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

The planning problem in MDPs is to compute an optimal policy. Classical approaches using an explicit representation of a model of size N are 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 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 [21], 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 [4]. 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 [4] 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 [2]. 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 [5, 16] 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 [10, 17]. In [12], 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.

2.4 The Chernoff-Hoeffding bounds for sampling

In section 3, we will use Chernoff-Hoeffding bounds [13] 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. *[13] 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.,* $N := \ln(\frac{2}{\delta})/2\varepsilon^2$. For instance, Chernoff-Hoeffding bounds are used in [6, 7] to obtain statistics representations to approximate automata runs and MDPs traces.

3. APPROXIMATE PLANNING AND PROBABILISTIC VERIFICATION

Taking into account that an approach with an explicit representation of an MDP is clearly infeasible for very large or infinite state spaces, we propose an efficient approximation method for planning and verification when an 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 learning 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. A randomized approximation scheme is a randomized algorithm which takes also as inputs two real parameters $\varepsilon > 0$ and $\delta < 1$. ε measures the approximation quality and $(1 - \delta)$ the confidence degree in the result of the randomized algorithm. For simplicity, we describe the two phases separately, but they are combined in the resulting algorithm.

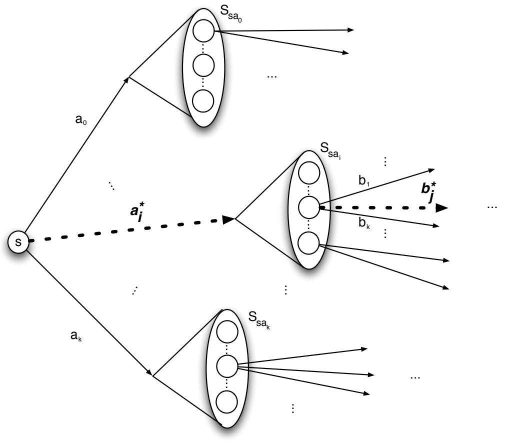

Figure 1: Structure of $\widehat{\mathcal{M}}$

3.1 Planning problem and Learning in MDPs

The learning phase of the verification algorithm is an adaptation of Kearns's learning algorithm [15]. 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, their algorithm uses the probabilistic generator to find a near-optimal action to perform from state s. The basic idea is to sample the probabilistic generator from states in the neighborhood of s.

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:

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

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

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

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.

3.2 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 [17] the authors show that the satisfaction probability of monotone or anti-monotone linear time properties can be approximated with a randomized approximation scheme. 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, they adapt the notion of randomized approximation scheme for counting problems, which is due to Karp and Luby [14], to the problem of computing probabilities. A probability problem is, 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. *[17] 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.

The authors of [17] use the probabilistic generator 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 $\mathcal{GAA}$ uses the probabilistic generator G to compute a good approximation of $Prob_k[\psi]$.

```
Generic approximation algorithm $\mathcal{GAA}$
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
Generate a random path $\sigma$ of length $k$
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 [13] on the tail of the distribution of a sum of independent random variables.

Theorem 1. *[17] 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 first in the APMC tool (see [11], and [8, 9] for distributed versions) , then in PRISM (see [12]).

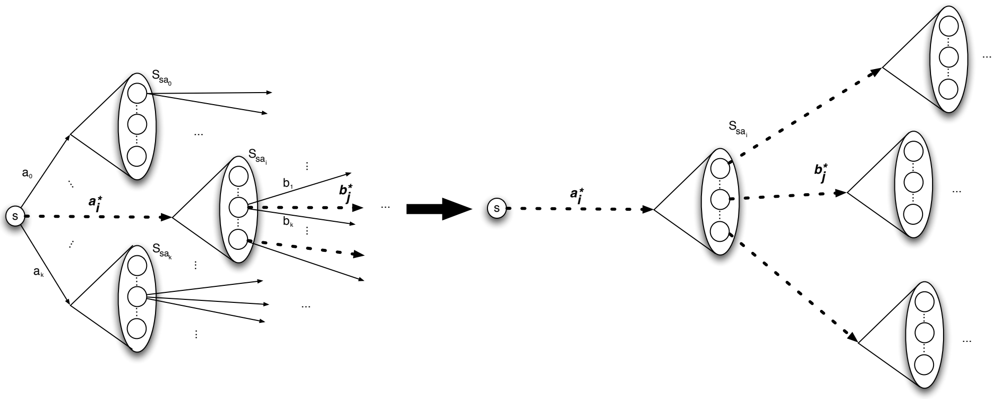

Figure 2: Step 3 of the $\mathcal{AVA}$ algorithm

3.3 Approximation algorithm for planning and verification in MDPs

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 $Prob[\psi|\widehat{\pi}^\star]$.

```
Approximate planning and verification algorithm APVA
Input: $G, \gamma, s, \psi, \varepsilon, \delta$
Output: $\varepsilon$-approximation of $Prob[\psi|\widehat{\pi}^\star]$
1. Compute horizon H and width C as functions of $\varepsilon, \gamma$ and 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 $(\mathcal{M}^*, H, \psi, \varepsilon, \delta)$
```

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 2 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 [15] 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 2. *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 $\geq 1 - \lambda/R_{max}$,*
- *the running time is $O((kC)^H)$ and the space complexity is $O(C^H)$.*

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

In the case where the discount factor γ is very close to 1, the use of another technique such as policy rollout (see section 4) can be envisioned as a future work.

4. RELATED WORK

There are two types of sampling methods to approximate planning problem in very large MDPs. The first one is the sparse sampling technique initiated by Kearns *et al.* (see [15]) and the planning phase of our 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 [20, 1]). 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.

5. CONCLUSION

We presented a first try for the design of an efficient randomized approximation method to combine planning and verification for very large or infinite 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.

6. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This research was funded by the ANR projects VERAP (ANR-07-SESU-013) and SPADES (08-ANR-SEGI-025).

7. REFERENCES

[1] D. Bertsekas and D. Castanon. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, 1999.

[2] A. Bianco and L. 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.

[3] H. Chang, M. Fu, J. Hu, and S. Marcus. A survey of some simulation-based algorithms for markov decision processes. *Communications in information and systems*, 7(1):59–92, 2007.

[4] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM (JACM)*, 42(4):857–907, 1995.

[5] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. 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.

[6] M. de Rougemont and M. Tracol. Statistic analysis for probabilistic processes. In *Proc. of the 24th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 299–308. IEEE Computer Society, 2009.

[7] E. Fischer, F. Magniez, and M. de Rougemont. Approximate Satisfiability and Equivalence. In *Proc. of the 21th IEEE Symposium on Logic in Computer Science (LICS)*, pages 421–430. IEEE Computer Society, 2006.

[8] G. Guirado, T. Hérault, R. Lassaigne, and S. 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.

[9] K. Hamidouche, A. Borghi, P. Esterie, J. Falcou, and S. Peyronnet. Three high performance architectures in the parallel approximate probabilistic model checking boat. In *Proc. of Parallel and Distributed Model Checking (PDMC)*, 2010.

[10] T. Herault, R. Lassaigne, F. Magniette, and S. 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.

[11] T. Herault, R. Lassaigne, and S. Peyronnet. APMC 3.0: Approximate verification of discrete and continuous time markov chains. In *Third International Conference on the Quantitative Evaluation of Systems (QEST)*, pages 129–130. IEEE Computer Society, 2006.

[12] A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. 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.

[13] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[14] R. M. Karp and M. 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.

[15] M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.

[16] M. Kwiatkowska, G. Norman, and D. 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.

[17] R. Lassaigne and S. Peyronnet. Probabilistic verification and approximation. *Annals of Pure and Applied Logic*, 152(1-3):122 – 131, 2008.

[18] M. L. Putterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley and Sons, 1994.

[19] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[20] G. Tesauro and G. Galperin. On-line policy improvement using monte-carlo search. *Advances in Neural Information Processing Systems*, pages 1068–1074, 1997.

[21] M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. of the 26th Foundations of Computer Science (FOCS)*, pages 327–338, 1984.