Verification of Markov Decision Processes using Learning Algorithms*

Tomáš Brázdil[1], Krishnendu Chatterjee[2], Martin Chmelík[2], Vojtěch Forejt[3], Jan Křetínský[2], Marta Kwiatkowska[3], David Parker[4], and Mateusz Ujma[3]

[1] Masaryk University, Brno, Czech Republic [2] IST Austria
[3] University of Oxford, UK [4] University of Birmingham, UK

Abstract. We present a general framework for applying machine-learning algorithms to the verification of Markov decision processes (MDPs). The primary goal of these techniques is to improve performance by avoiding an exhaustive exploration of the state space. Our framework focuses on probabilistic reachability, which is a core property for verification, and is illustrated through two distinct instantiations. The first assumes that full knowledge of the MDP is available, and performs a heuristic-driven partial exploration of the model, yielding precise lower and upper bounds on the required probability. The second tackles the case where we may only sample the MDP, and yields probabilistic guarantees, again in terms of both the lower and upper bounds, which provides efficient stopping criteria for the approximation. The latter is the first extension of statistical model checking for unbounded properties in MDPs. In contrast with other related techniques, our approach is not restricted to time-bounded (finite-horizon) or discounted properties, nor does it assume any particular properties of the MDP. We also show how our methods extend to LTL objectives. We present experimental results showing the performance of our framework on several examples.

1 Introduction

Markov decision processes (MDPs) are a widely used model for the formal verification of systems that exhibit stochastic behaviour. This may arise due to the possibility of failures (e.g. of physical system components), unpredictable events (e.g. messages sent across a lossy medium), or uncertainty about the environment (e.g. unreliable sensors in a robot). It may also stem from the explicit use of randomisation, such as probabilistic routing in gossip protocols or random back-off in wireless communication protocols.

Verification of MDPs against temporal logics such as PCTL and LTL typically reduces to the computation of optimal (minimum or maximum) reachability probabilities, either on the MDP itself or its product with some deterministic ω-automaton. Optimal reachability probabilities (and a corresponding optimal strategy for the MDP) can be computed in polynomial time through a reduction to linear programming, although in

* This research was funded in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM), 246967 (VERIWARE) and 279307 (Graph Games), by the EU FP7 project HIERATIC, by the Austrian Science Fund (FWF) projects S11402-N23 (RiSE), S11407-N23 (RiSE) and P23499-N23, by the Czech Science Foundation grant No P202/12/P612, by EPSRC project EP/K038575/1 and by the Microsoft faculty fellows award.

practice verification tools often use dynamic programming techniques, such as value iteration which approximates the values up to some pre-specified convergence criterion.

The efficiency or feasibility of verification is often limited by excessive time or space requirements, caused by the need to store a full model in memory. Common approaches to tackling this include: symbolic model checking, which uses efficient data structures to construct and manipulate a compact representation of the model; abstraction refinement, which constructs a sequence of increasingly precise approximations, bypassing construction of the full model using decision procedures such as SAT or SMT; and statistical model checking [38,19], which uses Monte Carlo simulation to generate approximate results of verification that hold with high probability.

In this paper, we explore the opportunities offered by learning-based methods, as used in fields such as planning or reinforcement learning [37]. In particular, we focus on algorithms that explore an MDP by generating trajectories through it and, whilst doing so, produce increasingly precise approximations for some property of interest (in this case, reachability probabilities). The approximate values, along with other information, are used as heuristics to guide the model exploration so as to minimise the solution time and the portion of the model that needs to be considered.

We present a general framework for applying such algorithms to the verification of MDPs. Then, we consider two distinct instantiations that operate under different assumptions concerning the availability of knowledge about the MDP, and produce different classes of results. We distinguish between *complete information*, where full knowledge of the MDP is available (but not necessarily generated and stored), and *limited information*, where (in simple terms) we can only sample trajectories of the MDP.

The first algorithm assumes complete information and is based on *real-time dynamic programming* (RTDP) [3]. In its basic form, this only generates approximations in the form of lower bounds (on maximum reachability probabilities). While this may suffice in some scenarios (e.g. planning), in the context of verification we typically require more precise guarantees. So we consider bounded RTDP (BRTDP) [31], which supplements this with an additional upper bound. The second algorithm assumes limited information and is based on *delayed Q-learning* (DQL) [36]. Again, we produce both lower and upper bounds but, in contrast to BRTDP, where these are guaranteed to be correct, DQL offers probably approximately correct (PAC) results, i.e., there is a non-zero probability that the bounds are incorrect.

Typically, MDP solution methods based on learning or heuristics make assumptions about the structure of the model. For example, the presence of end components [15] (subsets of states where it is possible to remain indefinitely with probability 1) can result in convergence to incorrect values. Our techniques are applicable to arbitrary MDPs. We first handle the case of MDPs that contain no end components (except for trivial designated goal or sink states). Then, we adapt this to the general case by means of *on-the-fly* detection of end components, which is one of the main technical contributions of the paper. We also show how our techniques extend to LTL objectives and thus also to minimum reachability probabilities.

Our DQL-based method, which yields PAC results, can be seen as an instance of statistical model checking [38,19], a technique that has received considerable attention. Until recently, most work in this area focused on purely probabilistic models, without

nondeterminism, but several approaches have now been presented for statistical model checking of nondeterministic models [13,14,27,4,28,18,29]. However, these methods all consider either time-bounded properties or use discounting to ensure convergence (see below for a summary). The techniques in this paper are the first for statistical model checking of unbounded properties on MDPs.

We have implemented our framework within the PRISM tool [25]. This paper concludes with experimental results for an implementation of our BRTDP-based approach that demonstrate considerable speed-ups over the fastest methods in PRISM.

Detailed proofs omitted due to lack of space are available in [7].

1.1 Related Work

In fields such as planning and artificial intelligence, many learning-based and heuristic-driven solution methods for MDPs have been developed. In the *complete information* setting, examples include RTDP [3] and BRTDP [31], as discussed above, which generate lower and lower/upper bounds on values, respectively. Most algorithms make certain assumptions in order to ensure convergence, for example through the use of a discount factor or by restricting to so-called Stochastic Shortest Path (SSP) problems, whereas we target arbitrary MDPs without discounting. More recently, an approach called FRET [24] was proposed for a generalisation of SSP, but this gives only a one-sided (lower) bound. We are not aware of any attempts to apply or adapt such methods in the context of probabilistic verification. A related paper is [1], which applies heuristic search methods to MDPs, but for generating probabilistic counterexamples.

As mentioned above, in the *limited information* setting, our algorithm based on delayed Q-learning (DQL) yields PAC results, similar to those obtained from *statistical model checking* [38,19,35]. This is an active area of research with a variety of tools [21,8,6,5]. In contrast with our work, most techniques focus on time-bounded properties, e.g., using bounded LTL, rather than *unbounded* properties. Several approaches have been proposed to transform checking of unbounded properties into testing of bounded properties, for example, [39,17,34,33]. However, these focus on purely probabilistic models, without nondeterminism, and do not apply to MDPs. In [4], unbounded properties are analysed for MDPs with spurious nondeterminism, where the way it is resolved does not affect the desired property.

More generally, the development of statistical model checking techniques for probabilistic models with *nondeterminism*, such as MDPs, is an important topic, treated in several recent papers. One approach is to give the nondeterminism a probabilistic semantics, e.g., using a uniform distribution instead, as for timed automata in [13,14,27]. Others [28,18], like this paper, aim to quantify over all strategies and produce an ε-optimal strategy. The work in [28] and [18] deals with the problem in the setting of discounted (and for the purposes of approximation thus bounded) or bounded properties, respectively. In the latter work, candidates for optimal schedulers are generated and gradually improved, but "at any given point we cannot quantify how close to optimal the candidate scheduler is" (cited from [18]) and the algorithm "does not in general converge to the true optimum" (cited from [30]). Further, [30] considers compact representation of schedulers, but again focuses only on (time) bounded properties.

Since statistical model checking is simulation-based, one of the most important difficulties is the analysis of *rare events*. This issue is, of course, also relevant for our

approach; see the section on experimental results. Rare events have been addressed using methods such as importance sampling [17,20] and importance splitting [22].

End components in MDPs can be collapsed either for algorithmic correctness [15] or efficiency [11] (where only lower bounds on maximum reachability probabilities are considered). Asymptotically efficient ways to detect them are given in [10,9].

2 Basics about MDPs and Learning Algorithms

We begin with basic background material on MDPs and some fundamental definitions for our learning framework. We use $\mathbb{N}$, $\mathbb{Q}$, and $\mathbb{R}$ to denote the sets of all non-negative integers, rational numbers and real numbers respectively. $Dist(X)$ is the set of all rational probability distributions over a finite or countable set X, i.e., the functions $f : X \to [0,1] \cap \mathbb{Q}$ such that $\sum_{x \in X} f(x) = 1$, and $supp(f)$ denotes the *support* of f.

2.1 Markov Decision Processes

We work with *Markov decision processes* (MDPs), a widely used model to capture both nondeterminism (e.g., for control or concurrency) and probability.

Definition 1. *An* MDP *is a tuple* $\mathsf{M} = \langle S, \overline{s}, A, E, \Delta \rangle$*, where* S *is a finite set of* states*,* $\overline{s} \in S$ *is an* initial state*,* A *is a finite set of* actions*,* $E : S \to 2^A$ *assigns non-empty sets of* enabled *actions to all states, and* $\Delta : S \times A \to Dist(S)$ *is a (partial)* probabilistic transition function *defined for all* s *and* a *where* $a \in E(s)$.

Remark 1. For simplicity of presentation we assume w.l.o.g. that, for every action $a \in A$, there is at most one state s such that $a \in E(s)$, i.e., $E(s) \cap E(s') = \emptyset$ for $s \neq s'$. If there *are* states s, s' such that $a \in E(s) \cap E(s')$, we can always rename the actions as $(s,a) \in E(s)$, and $(s',a) \in E(s')$, so that the MDP satisfies our assumption.

An *infinite path* of an MDP M is an infinite sequence $\omega = s_0 a_0 s_1 a_1 \ldots$ such that $a_i \in E(s_i)$ and $\Delta(s_i, a_i)(s_{i+1}) > 0$ for every $i \in \mathbb{N}$. A *finite path* is a finite prefix of an infinite path ending in a state. We use $last(\omega)$ to denote the last state of a finite path ω. We denote by $IPath$ (resp. $FPath$) the set of all infinite (resp. finite) paths, and by $IPath_s$ (resp. $FPath_s$) the set of infinite (resp. finite) paths starting in a state s.

A state s is *terminal* if all actions $a \in E(s)$ satisfy $\Delta(s,a)(s) = 1$. An *end component* (EC) of M is a pair (S', A') where $S' \subseteq S$ and $A' \subseteq \bigcup_{s \in S'} E(s)$ such that: (1) if $\Delta(s,a)(s') > 0$ for some $s \in S'$ and $a \in A'$, then $s' \in S'$; and (2) for all $s, s' \in S'$ there is a path $\omega = s_0 a_0 \ldots s_n$ such that $s_0 = s$, $s_n = s'$ and for all $0 \leq i < n$ we have $a_i \in A'$. A *maximal end component* (MEC) is an EC that is maximal with respect to the point-wise subset ordering.

Strategies. A *strategy* of MDP M is a function $\sigma : FPath \to Dist(A)$ satisfying $supp(\sigma(\omega)) \subseteq E(last(\omega))$ for every $\omega \in FPath$. Intuitively, the strategy resolves the choices of actions in each finite path by choosing (possibly at random) an action enabled in the last state of the path. We write Σ_{M} for the set of all strategies in M. In standard fashion [23], a strategy σ induces, for any initial state s, a probability measure $Pr^{\sigma}_{\mathsf{M},s}$ over $IPath_s$. A strategy σ is *memoryless* if $\sigma(\omega)$ depends only on $last(\omega)$.

Objectives and values. Given a set $F \subseteq S$ of target states, *bounded reachability* for step k, denoted by $\Diamond^{\leq k} F$, refers to the set of all infinite paths that reach a state in F within k steps, and *unbounded reachability*, denoted by $\Diamond F$, refers to the set of all infinite paths that reach a state in F. Note that $\Diamond F = \bigcup_{k \geq 0} \Diamond^{\leq k} F$. We consider the *reachability probability* $Pr^{\sigma}_{\mathsf{M},s}(\Diamond F)$, and strategies that maximise this probability. We denote by $V(s)$ the *value* in s, defined by $\sup_{\sigma \in \Sigma_{\mathsf{M}}} Pr^{\sigma}_{\mathsf{M},s}(\Diamond F)$. Given $\varepsilon \geq 0$, we say that a strategy σ is *ε-optimal* in s if $Pr^{\sigma}_{\mathsf{M},s}(\Diamond F) + \varepsilon \geq V(s)$, and we call a 0-optimal strategy *optimal*. It is known [32] that, for every MDP and set F, there is a memoryless optimal strategy for $\Diamond F$. We are interested in strategies that approximate the value function, i.e., ε-optimal strategies for some $\varepsilon > 0$.

2.2 Learning Algorithms for MDPs

In this paper, we study a class of learning-based algorithms that stochastically approximate the value function of an MDP. Let us fix, for this section, an MDP $\mathsf{M} = \langle S, \overline{s}, A, E, \Delta \rangle$ and target states $F \subseteq S$. We denote by $V : S \times A \to [0,1]$ the *value function* for state-action pairs of M, defined for all (s,a) where $s \in S$ and $a \in E(s)$:

$$V(s,a) := \sum\nolimits_{s' \in S} \Delta(s,a)(s') \cdot V(s').$$

Intuitively, $V(s,a)$ is the value in s assuming that the first action performed is a. A *learning algorithm* $\mathcal{A}$ simulates executions of M, and iteratively updates upper and lower approximations $U : S \times A \to [0,1]$ and $L : S \times A \to [0,1]$, respectively, of the value function $V : S \times A \to [0,1]$.

The functions U and L are initialised to appropriate values so that $L(s,a) \leq V(s,a) \leq U(s,a)$ for all $s \in S$ and $a \in A$. During the computation of $\mathcal{A}$, simulated executions start in the initial state $\overline{s}$ and move from state to state according to choices made by the algorithm. The values of $U(s,a)$ and $L(s,a)$ are updated for the states s visited by the simulated execution. Since $\max_{a \in E(s)} U(s,a)$ and $\max_{a \in E(s)} L(s,a)$ represent upper and lower bound on $V(s)$, a learning algorithm $\mathcal{A}$ terminates when $\max_{a \in E(\overline{s})} U(\overline{s},a) - \max_{a \in E(\overline{s})} L(\overline{s},a) < \varepsilon$ where the *precision* $\varepsilon > 0$ is given to the algorithm as an argument. Note that, because U and L are possibly updated based on the simulations, the computation of the learning algorithm may be randomised and even give incorrect results with some probability.

Definition 2. *Denote by $\mathcal{A}(\varepsilon)$ the instance of learning algorithm $\mathcal{A}$ with precision ε. We say that $\mathcal{A}$* converges surely (resp. almost surely) *if, for every $\varepsilon > 0$, the computation of $\mathcal{A}(\varepsilon)$ surely (resp. almost surely) terminates, and $L(\overline{s},a) \leq V(\overline{s},a) \leq U(\overline{s},a)$ holds upon termination.*

In some cases, almost-sure convergence cannot be guaranteed, so we demand that the computation terminates correctly with sufficiently high probability. In such cases, we assume the algorithm is also given a *confidence* $\delta > 0$ as an argument.

Definition 3. *Denote by $\mathcal{A}(\varepsilon, \delta)$ the instance of learning algorithm $\mathcal{A}$ with precision ε and confidence δ. We say that $\mathcal{A}$ is* probably approximately correct (PAC) *if, for every $\varepsilon > 0$ and every $\delta > 0$, with probability at least $1 - \delta$, the computation of $\mathcal{A}(\varepsilon, \delta)$ terminates with $L(\overline{s},a) \leq V(\overline{s},a) \leq U(\overline{s},a)$.*

The function U defines a memoryless strategy σ_U which in every state s chooses all actions a maximising the value $U(s,a)$ over $E(s)$ uniformly at random. The strategy σ_U is used in some of the algorithms and also contributes to the output.

Remark 2. If the value function is defined as the infimum over strategies (as in [31]), then the strategy chooses actions to minimise the lower value. Since we consider the dual case of supremum over strategies, the choice of σ_U is to maximise the upper value.

We also need to specify what knowledge about the MDP M is available to the learning algorithm. We distinguish the following two distinct cases.

Definition 4. *A learning algorithm has* limited information *about* M *if it knows only the initial state* $\overline{s}$*, a number* $K \geq |S|$*, a number* $E_m \geq \max_{s \in S} |E(s)|$*, a number* $0 < q \leq p_{\min}$*, where* $p_{\min} = \min\{\Delta(s,a)(s') \mid s \in S, a \in E(s), s' \in supp(\Delta(s,a))\}$*, and the function* E *(more precisely, given a state* s*, the learning procedure can ask an oracle for* $E(s)$*). We assume that the algorithm may simulate an execution of* M *starting with* $\overline{s}$ *and choosing enabled actions in individual steps.*

Definition 5. *A learning algorithm has* complete information *about* M *if it knows the complete MDP* M*.*

Note that the MDPs we consider are "fully observable", so even in the limited information case strategies can make decisions based on the precise state of the system.

3 MDPs without End Components

We first present algorithms for MDPs *without* ECs, which considerably simplifies the adaptation of BRTDP and DQL to unbounded reachability objectives. Later, in Section 4, we extend our methods to deal with arbitrary MDPs (*with* ECs). Let us fix an MDP $\mathsf{M} = \langle S, \overline{s}, A, E, \Delta \rangle$ and a target set F. Formally, we assume the following.

Assumption-EC. MDP M has no ECs, except two trivial ones containing distinguished terminal states 1 and 0, respectively, with $F = \{1\}$, $V(1) = 1$ and $V(0) = 0$.

3.1 Our framework

We start by formalising a general framework for learning algorithms, as outlined in the previous section. We then instantiate this and obtain two learning algorithms: BRTDP and DQL. Our framework is presented as Algorithm 1, and works as follows. Recall that functions U and L store the current upper and lower bounds on the value function V. Each iteration of the outer loop is divided into two phases: EXPLORE and UPDATE. In the EXPLORE phase (lines 5 - 10), the algorithm samples a finite path ω in M from $\overline{s}$ to a state in $\{1, 0\}$ by always randomly choosing one of the enabled actions that maximises the U value, and sampling the successor state using the probabilistic transition function. In the UPDATE phase (lines 11 - 16), the algorithm updates U and L on the state-action pairs along the path in a backward manner. Here, the function *pop* pops and returns the last letter of the given sequence.

Algorithm 1 Learning algorithm (for MDPs with no ECs)

Inputs: An EC-free MDP M
$U(\cdot,\cdot) \leftarrow 1, L(\cdot,\cdot) \leftarrow 0$
$L(1,\cdot) \leftarrow 1, U(0,\cdot) \leftarrow 0$ ▷ INITIALISE
repeat
  $\omega \leftarrow \overline{s}$ /* EXPLORE phase */
  repeat
    $a \leftarrow$ sampled uniformly from $\arg\max_{a \in E(last(\omega))} U(last(\omega), a)$
    $s \leftarrow$ sampled according to $\Delta(last(\omega), a)$ ▷ GETSUCC(ω, a)
    $\omega \leftarrow \omega\, a\, s$
  until $s \in \{1, 0\}$ ▷ TERMINATEPATH(ω)
  repeat /* UPDATE phase */
    $s' \leftarrow pop(\omega)$
    $a \leftarrow pop(\omega)$
    $s \leftarrow last(\omega)$
    UPDATE$((s, a), s')$
  until $\omega = \overline{s}$
until $\max_{a \in E(\overline{s})} U(\overline{s}, a) - \max_{a \in E(\overline{s})} L(\overline{s}, a) < \varepsilon$ ▷ TERMINATE

3.2 Instantiations: BRTDP and DQL

Our two algorithm instantiations, BRTDP and DQL, differ in the definition of UPDATE.

Unbounded reachability with BRTDP. We obtain BRTDP by instantiating UPDATE with Algorithm 2, which requires complete information about the MDP. Intuitively, UPDATE computes new values of $U(s, a)$ and $L(s, a)$ by taking the weighted average of the corresponding U and L values, respectively, over all successors of s via action a. Formally, denote $U(s) = \max_{a \in E(s)} U(s, a)$ and $L(s) = \max_{a \in E(s)} L(s, a)$.

Algorithm 2 BRTDP instantiation of Algorithm 1

procedure UPDATE$((s, a), \cdot)$
  $U(s, a) := \sum_{s' \in S} \Delta(s, a)(s') U(s')$
  $L(s, a) := \sum_{s' \in S} \Delta(s, a)(s') L(s')$

The following theorem says that BRTDP satisfies the conditions of Definition 2 and never returns incorrect results.

Theorem 1. *The algorithm BRTDP converges almost surely under Assumption-EC.*

Remark 3. Note that, in the EXPLORE phase, an action maximising the value of U is chosen and the successor is sampled according to the probabilistic transition function of M. However, we can consider various modifications. Actions and successors may be chosen in different ways (e.g., for GETSUCC), for instance, uniformly at random, in a round-robin fashion, or assigning various probabilities (bounded from below by some fixed $p > 0$) to all possibilities in any biased way. In order to guarantee almost-sure convergence, some conditions have to be satisfied. Intuitively we require, that the state-action pairs used by ε-optimal strategies have to be chosen enough times. If this condition is satisfied then the almost-sure convergence is preserved and the practical running times may significantly improve. For details, see Section 5.

Remark 4. The previous BRTDP algorithm is only applicable if the transition probabilities are known. However, if complete information is not known, but $\Delta(s,a)$ can be repeatedly sampled for any s and a, then a variant of BRTDP can be shown to be probably approximately correct.

Unbounded reachability with DQL. Often, complete information about the MDP is unavailable, repeated sampling is not possible, and we have to deal with only limited information about M (see Definition 4). For this scenario, we use DQL, which can be obtained by instantiating UPDATE with Algorithm 3.

Algorithm 3 DQL (delay m, estimator precision $\bar{\varepsilon}$) instantiation of Algorithm 1

```
1:  procedure UPDATE((s,a), s')
2:      if c(s,a) = m and LEARN(s,a) then
3:          if accum^U_m(s,a)/m < U(s,a) − 2ε̄ then
4:              U(s,a) ← accum^U_m(s,a)/m + ε̄
5:              accum^U_m(s,a) = 0
6:          if accum^L_m(s,a)/m > L(s,a) + 2ε̄ then
7:              L(s,a) ← accum^L_m(s,a)/m − ε̄
8:              accum^L_m(s,a) = 0
9:          c(s,a) = 0
10:     else
11:         accum^U_m(s,a) ← accum^U_m(s,a) + U(s')
12:         accum^L_m(s,a) ← accum^L_m(s,a) + L(s')
13:         c(s,a) ← c(s,a) + 1
```

Macro LEARN(s,a) is true in the kth call of UPDATE$((s,a),\cdot)$ if, since the $(k-2m)$th call of UPDATE$((s,a),\cdot)$, line 4 was not executed in any call of UPDATE$(\cdot,\cdot)$.

The main idea behind DQL is as follows. As the probabilistic transition function is not known, we cannot update $U(s,a)$ and $L(s,a)$ with the actual values $\sum_{s'\in S}\Delta(s,a)(s')U(s')$ and $\sum_{s'\in S}\Delta(s,a)(s')L(s')$, respectively. However, we can instead use simulations executed in the EXPLORE phase of Algorithm 1 to estimate these values. Namely, we use $accum^U_m(s,a)/m$ to estimate $\sum_{s'\in S}\Delta(s,a)(s')U(s')$ where $accum^U_m(s,a)$ is the sum of the U values of the last m immediate successors of (s,a) seen during the EXPLORE phase. Note that the delay m must be chosen large enough for the estimates to be sufficiently close, i.e., $\bar{\varepsilon}$-close, to the real values.

So, in addition to $U(s,a)$ and $L(s,a)$, the algorithm uses new variables $accum^U_m(s,a)$ and $accum^L_m(s,a)$ to accumulate $U(s,a)$ and $L(s,a)$ values, respectively, and a counter $c(s,a)$ recording the number of invocations of a in s since the last update (all these variables are initialised to 0 at the beginning of computation). Assume that a has been invoked in s during the EXPLORE phase of Algorithm 1, which means that UPDATE$((s,a),s')$ is eventually called in the UPDATE phase of Algorithm 1 with the corresponding successor s' of (s,a). If $c(s,a)=m$ at that time, a has been invoked in s precisely m times since the last update concerning (s,a) and the procedure UPDATE$((s,a),s')$ updates $U(s,a)$ with $accum^U_m(s,a)/m$ plus an appropriate constant $\bar{\varepsilon}$ (unless LEARN is false). Here, the purpose of adding $\bar{\varepsilon}$ is to make $U(s,a)$ stay above the real value $V(s,a)$ with high probability. If $c(s,a)<m$, then

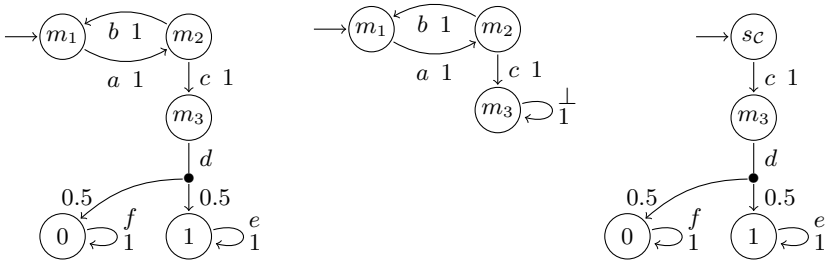

Fig. 1. MDP M with an EC (left), MDP $\mathsf{M}^{\{m_1,m_2\}}$ constructed from M in on-the-fly BRTDP (centre), and MDP M' obtained from M by collapsing $\mathcal{C} = (\{m_1, m_2\}, \{a, b\})$ (right).

UPDATE$((s,a),s')$ simply accumulates $U(s')$ into $accum_m^U(s,a)$ and increases the counter $c(s,a)$. The $L(s,a)$ values are estimated by $accum_m^L(s,a)/m$ in a similar way, just subtracting $\bar{\varepsilon}$ from $accum_m^L(s,a)$. The procedure requires m and $\bar{\varepsilon}$ as inputs, and they are chosen depending on ε and δ; more precisely, we choose $\bar{\varepsilon} = \frac{\varepsilon \cdot (p_{\min}/E_m)^{|S|}}{12|S|}$ and $m = \frac{\ln(6|S||A|(1+\frac{|S||A|}{\bar{\varepsilon}})/\delta)}{2\bar{\varepsilon}^2}$ and establish that DQL is probably approximately correct. The parameters m and $\bar{\varepsilon}$ can be conservatively approximated using only the limited information about the MDP (i.e. using K, E_m and q). Even though the algorithm has limited information about M, we still establish the following theorem.

Theorem 2. *DQL is probably approximately correct under Assumption-EC.*

Bounded reachability. Algorithm 1 can be trivially adapted to handle bounded reachability properties by preprocessing the input MDP in standard fashion. Namely, every state is equipped with a bounded counter with values ranging from 0 to k where k is the step bound, the current value denoting the number of steps taken so far. All target states remain targets for all counter values, and every non-target state with counter value k becomes rejecting. Then, to determine the k-step reachability in the original MDP, we compute the (unbounded) reachability in the new MDP. Although this means that the number of states is multiplied by $k+1$, in practice the size of the explored part of the model can be small.

4 Unrestricted MDPs

We first illustrate with an example that the algorithms BRTDP and DQL as presented in Section 3 may not converge when there are ECs in the MDP.

Example 1. Consider the MDP M in Fig. 1 (left) with EC $(\{m_1, m_2\}, \{a, b\})$. The values in states m_1, m_2 are $V(m_1) = V(m_2) = 0.5$ but the upper bounds are $U(m_1) = U(m_2) = 1$ for every iteration. This is because $U(m_1, a) = U(m_2, b) = 1$ and both algorithms greedily choose the action with the highest upper bound. Thus, in every iteration t of the algorithm, the error for the initial state m_1 is $U(m_1) - V(m_1) = \frac{1}{2}$ and the algorithm does not converge. In general, any state in an EC has upper bound 1

since, by definition, there are actions that guarantee the next state is in the EC, i.e., is a state with upper bound 1. This argument holds even for standard value iteration with values initialised to 1.

One way of dealing with general MDPs is to preprocess them to identify all MECs [10,9] and "collapse" them into single states (see e.g. [15,11]). These algorithms require that the graph model is known and explore the whole state space, but this may not be possible either due to limited information (see Definition 4) or because the model is too large. Hence, we propose a modification to the algorithms from the previous sections that allows us to deal with ECs "on-the-fly". We first describe the collapsing of a set of states and then present a crucial lemma that allows us to identify ECs to collapse.

Collapsing states. In the following, we say that an MDP $\mathsf{M}' = \langle S', \overline{s}', A', E', \Delta' \rangle$ is obtained from $\mathsf{M} = \langle S, \overline{s}, A, E, \Delta \rangle$ by *collapsing* a tuple (R, B), where $R \subseteq S$ and $B \subseteq A$ with $B \subseteq \bigcup_{s \in R} E(s)$ if:

- $S' = (S \setminus R) \cup \{s_{(R,B)}\}$,
- $\overline{s}'$ is either $s_{(R,B)}$ or $\overline{s}$, depending on whether $\overline{s} \in R$ or not,
- $A' = A \setminus B$,
- $E'(s) = E(s)$, for $s \in S \setminus R$; $\quad E'(s_{(R,B)}) = \bigcup_{s \in R} E(s) \setminus B$,
- Δ' is defined for all $s \in S'$ and $a \in E'(s)$ by:
  - $\Delta'(s,a)(s') = \Delta(s,a)(s')$ for $s, s' \neq s_{(R,B)}$,
  - $\Delta'(s,a)(s_{(R,B)}) = \sum_{s' \in R} \Delta(s,a)(s')$ for $s \neq s_{(R,B)}$,
  - $\Delta'(s_{(R,B)},a)(s') = \Delta(s,a)(s')$ for $s' \neq s_{(R,B)}$ and s the unique state with $a \in E(s)$ (see Remark 1),
  - $\Delta'(s_{(R,B)},a)(s_{(R,B)}) = \sum_{s' \in R} \Delta(s,a)(s')$ where s is the unique state with $a \in E(s)$.

We denote the above transformation, which creates M' from M, as the COLLAPSE function, i.e., COLLAPSE(R, B). As a special case, given a state s and a terminal state $s' \in \{0, 1\}$, we use MAKETERMINAL(s, s') as shorthand for COLLAPSE$(\{s, s'\}, E(s))$, where the new state is renamed to s'. Intuitively, after MAKETERMINAL(s, s'), every transition previously leading to state s will now lead to the terminal state s'.

For practical purposes, it is important to note that the collapsing does not need to be implemented explicitly, but can be done by keeping a separate data structure which stores information about the collapsed states.

Identifying ECs from simulations. Our modifications will identify ECs "on-the-fly" through simulations that get stuck in them. The next lemma establishes the identification principle. To this end, for a path ω, let us denote by $Appear(\omega, i)$ the tuple (S_i, A_i) of M such that $s \in S_i$ and $a \in A_i(s)$ if and only if (s, a) occurs in ω more than i times.

Lemma 1. *Let* $c = \exp\left(-\left(p_{\min}/E_m\right)^{\kappa} / \kappa\right)$*, where* $\kappa = KE_m + 1$*, and let* $i \geq \kappa$*. Assume that the* EXPLORE *phase in Algorithm 1 terminates with probability less than* 1*. Then, provided the* EXPLORE *phase does not terminate within* $3i^3$ *iterations, the conditional probability that* $Appear(\omega, i)$ *is an EC is at least* $1 - 2c^i i^3 \cdot \left(p_{\min}/E_m\right)^{-\kappa}$.

The above lemma allows us to modify the EXPLORE phase of Algorithm 1 in such a way that simulations will be used to identify ECs. The ECs discovered will subsequently be collapsed. We first present the overall skeleton (Algorithm 4) for treating

ECs "on-the-fly", which consists of two parts: (i) identification of ECs; and (ii) processing them. The instantiations for BRTDP and DQL will differ in the identification phase. Hence, before proceeding to the individual identification algorithms, we first establish the correctness of the processing phase.

Algorithm 4 Extension for general MDPs

```
 1: function ON-THE-FLY-EC
 2:     M ← IDENTIFYECS                          ▷ IDENTIFICATION OF ECS
 3:     for all (R, B) ∈ M do                    ▷ PROCESS ECS
 4:         COLLAPSE(R, B)
 5:         for all s ∈ R and a ∈ E(s) \ B do
 6:             U(s_(R,B), a) ← U(s, a)
 7:             L(s_(R,B), a) ← L(s, a)
 8:         if R ∩ F ≠ ∅ then
 9:             MAKETERMINAL(s_(R,B), 1)
10:         else if no actions enabled in s_(R,B) then
11:             MAKETERMINAL(s_(R,B), 0)
```

Lemma 2. *Assume (R, B) is an EC in MDP M, V_{M} the value before the* PROCESS ECS *procedure in Algorithm 4, and $V_{\mathsf{M}'}$ the value after the procedure, then:*

- *for $i \in \{0, 1\}$ if* MAKETERMINAL$(s_{(R,B)}, i)$ *is called, then $\forall s \in R : V_{\mathsf{M}}(s) = i$,*
- $\forall s \in S \setminus R : V_{\mathsf{M}}(s) = V_{\mathsf{M}'}(s)$,
- $\forall s \in R : V_{\mathsf{M}}(s) = V_{\mathsf{M}'}(s_{(R,B)})$.

Interpretation of collapsing. Intuitively, once an EC (R, B) is collapsed, the algorithm in the EXPLORE phase can choose a state $s \in R$ and action $a \in E(s) \setminus B$ to leave the EC. This is simulated in the EXPLORE phase by considering all actions of the EC uniformly at random until s is reached, and then action a is chosen. Since (R, B) is an EC, playing all actions of B uniformly at random ensures s is almost surely reached. Note that the steps made inside a collapsed EC do not count towards the length of the explored path.

Now, we present the on-the-fly versions of BRTDP and DQL. For each case, we describe: (i) modification of Algorithm 1; (ii) identification of ECs; and (iii) correctness.

4.1 Complete information (BRTDP)

Modification of Algorithm 1. To obtain BRTDP working with unrestricted MDPs, we modify Algorithm 1 as follows: for iteration i of the EXPLORE phase, we insert a check after line 9 such that, if the length of the path ω explored (i.e., the number of states) is k_i (see below), then we invoke the ON-THE-FLY-EC function for BRTDP. The ON-THE-FLY-EC function possibly modifies the MDP by processing (collapsing) some ECs as described in Algorithm 4. After the ON-THE-FLY-EC function terminates, we interrupt the current EXPLORE phase, and start the EXPLORE phase for the $i+1$-th iteration (i.e., generating a new path again, starting from $\overline{s}$ in the modified MDP). To complete the description we describe the choice of k_i and identification of ECs.

Choice of k_i. Because computing ECs can be expensive, we do not call ON-THE-FLY-EC every time a new state is explored, but only after every k_i steps of the repeat-until

loop at lines 6–10 in iteration i. The specific value of k_i can be decided experimentally and change as the computation progresses. A theoretical bound for k_i to ensure that there is an EC with high probability can be obtained from Lemma 1.

Identification of ECs. Given the current explored path ω, let (T, G) be $Appear(\omega, 0)$, that is, the set of states and actions explored in ω. To obtain the ECs from the set T of explored states, we use Algorithm 5. This computes an auxiliary MDP $\mathsf{M}^T = \langle T', \overline{s}, A', E', \Delta' \rangle$ defined as follows:

- $T' = T \cup \{t \mid \exists s \in T, a \in E(s) \text{ such that } \Delta(s,a)(t) > 0\}$,
- $A' = \bigcup_{s \in T} E(s) \cup \{\bot\}$,
- $E'(s) = E(s)$ if $s \in T$ and $E'(s) = \{\bot\}$ otherwise,
- $\Delta'(s,a) = \Delta(s,a)$ if $s \in T$, and $\Delta'(s,\bot)(s) = 1$ otherwise.

It then computes all MECs of M^T that are contained in T and identifies them as ECs. The following lemma states that each of these is indeed an EC in the original MDP.

Algorithm 5 Identification of ECs for BRTDP

1: **function** IDENTIFYECS(M, T)
2: compute M^T
3: $\mathcal{M}' \leftarrow$ MECs of M^T
4: $\mathcal{M} \leftarrow \{(R,B) \in \mathcal{M}' \mid R \subseteq T\}$

Lemma 3. *Let M, M^T be the MDPs from the construction above and T be the set of explored states. Then every MEC (R,B) in M^T such that $R \subseteq T$ is an EC in M.*

Finally, we establish that the modified algorithm, which we refer to as *on-the-fly BRTDP*, almost surely converges; the proof is an extension of Theorem 1.

Theorem 3. *On-the-fly BRTDP converges almost surely for all MDPs.*

Example 2. Let us describe the execution of the on-the-fly BRTDP on the MDP M from Fig. 1 (left). Choose $k_i \geq 6$ for all i. The loop at lines 6 to 10 of Algorithm 1 generates a path ω that contains some (possibly zero) number of loops $m_1\, a\, m_2 b$ followed by $m_1\, a\, m_2\, c\, m_3\, d\, t$ where $t \in \{0,1\}$. In the subsequent UPDATE phase, we set $U(m_3,d) = L(m_3,d) = 0.5$ and then $U(m_2,c) = L(m_2,c) = 0.5$; none of the other values change. In the second iteration of the loop at lines 6 to 10, the path $\omega' = m_1\, a\, m_2\, b\, m_1\, a\, m_2\, b \ldots$ is being generated, and the newly inserted check for ON-THE-FLY-EC will be triggered once ω achieves the length k_i.

The algorithm now aims to identify ECs in the MDP based on the part of the MDP explored so far. To do so, the MDP M^T for the set $T = \{m_1, m_2\}$ is constructed and we depict it in Fig. 1 (centre). We then run MEC detection on M^T, finding that $(\{m_1, m_2\}, \{a, b\})$ is an EC, and so it gets collapsed according to the COLLAPSE procedure. This gives the MDP M' from Fig. 1 (right).

The execution then continues with M'. A new path is generated at lines 6 to 10 of Algorithm 1; suppose it is $\omega'' = s_{\mathcal{C}} c m_3 d 0$. In the UPDATE phase we then update the value $U(s_{\mathcal{C}}, d) = L(s_{\mathcal{C}}, d) = 0.5$, which makes the condition at the last line of Algorithm 1 satisfied, and the algorithm finishes, having computed the correct value.

4.2 Limited information (DQL)

Modification of Algorithm 1 and identification of ECs. The modification of Algorithm 1 is done exactly as for the modification of BRDTP (i.e., we insert a check after line 9 of EXPLORE, which invokes the ON-THE-FLY-EC function if the length of path ω exceeds k_i). In iteration i, we set k_i as $3\ell_i^3$, for some ℓ_i (to be described later). The identification of the EC is as follows: we consider $Appear(\omega, \ell_i)$, the set of states and actions that have appeared more than ℓ_i times in the explored path ω, which is of length $3\ell_i^3$, and identify the set as an EC; i.e., $\mathcal{M}$ in line 2 of Algorithm 4 is defined as the set containing the single tuple $Appear(\omega, \ell_i)$. We refer to the algorithm as *on-the-fly DQL*.

Choice of ℓ_i and correctness. The choice of ℓ_i is as follows. Note that, in iteration i, the error probability, obtained from Lemma 1, is at most $2c^{\ell_i}\ell_i^3 \cdot (p_{\min}/E_m)^{-\kappa}$ and we choose ℓ_i such that $2c^{\ell_i}\ell_i^3 \cdot (p_{\min}/E_m)^{-\kappa} \leq \frac{\delta/2}{2^i}$, where δ is the confidence. Note that, since $c < 1$, we have that c^{ℓ_i} decreases exponentially, and hence for every i such ℓ_i exists. It follows that the total error of the algorithm due to the on-the-fly EC collapsing is at most $\delta/2$. It follows from the proof of Theorem 2 that for on-the-fly DQL the error is at most δ if we use the same $\bar{\varepsilon}$ as for DQL, but now with DQL confidence $\delta/4$, i.e., with $m = \frac{\ln(24|S||A|(1+\frac{|S||A|}{\bar{\varepsilon}})/\delta)}{2\bar{\varepsilon}^2}$. As before, these numbers can be conservatively approximated using the limited information.

Theorem 4. *On-the-fly DQL is probably approximately correct for all MDPs.*

Example 3. Let us now briefly explain the execution of on-the-fly DQL on the MDP M from Fig. 1 (left). At first, paths of the same form as ω in Example 2 will be generated and there will be no change to U and L, because in any call to UPDATE (see Algorithm 3) for states $s \in \{m_1, m_2\}$ with $c(s, a) = m$ the values accumulated in $accum_m^U(s, a)/m$ and $accum_m^L(s, a)/m$ are the same as the values already held, namely 1 and 0, respectively.

At some point, we call UPDATE for the tuple (m_3, d) with $c(m_3, d) = m$, which will result in the change of $U(m_3, d)$ and $L(m_3, d)$. Note, that at this point, the numbers $accum_m^U(s, d)/m$ and $accum_m^L(s, d)/m$ are both equal to the proportion of generated paths that visited the state 1. This number will, with high probability, be very close to 0.5, say 0.499. We thus set $U(m_3, d) = 0.499 + \varepsilon$ and $L(m_3, d) = 0.499 - \varepsilon$.

We then keep generating paths of the same form and at some point also update $U(m_2, c)$ and $L(m_2, c)$ to precisely $0.499 + \varepsilon$ and $0.499 - \varepsilon$, respectively. The subsequently generated path will be looping on m_1 and m_2, and once it is of length ℓ_i, we identify $(\{m_1, m_2\}, \{a, b\})$ as an EC due to the definition of $Appear(\omega, \ell_i)$. We then get the MDP from Fig. 1 (right), which we use to generate new paths, until the upper and lower bounds on value in the new initial state are within the required bound.

4.3 Extension to LTL

So far we have focused on reachability, but our techniques also extend to linear temporal logic (LTL) objectives. By translating an LTL formula to an equivalent deterministic ω-automaton, verifying MDPs with LTL objectives reduces to analysis of MDPs with ω-regular conditions such as Rabin acceptance conditions. A Rabin acceptance condition consists of a set $\{(M_1, N_1) \dots (M_d, N_d)\}$ of d pairs (M_i, N_i), where each $M_i \subseteq S$ and

$N_i \subseteq S$. The acceptance condition requires that, for some $1 \leq i \leq d$, states in M_i are visited infinitely often and states in N_i are visited finitely often.

Value computation for MDPs with Rabin objectives reduces to optimal reachability of *winning* ECs, where an EC (R, B) is winning if $R \cap M_i \neq \emptyset$ and $R \cap N_i = \emptyset$ for some $1 \leq i \leq d$ [12]. Thus, extending our results from reachability to Rabin objectives requires processing of ECs for Rabin objectives (line 3-11 of Algorithm 4), which is done as follows. Once an EC (R, B) is identified, we first obtain the EC in the original MDP (i.e., obtain the set of states and actions corresponding to the EC in the original MDP) as $(\overline{R}, \overline{B})$ and then determine if there is a sub-EC of $(\overline{R}, \overline{B})$ that is winning using standard algorithms for MDPs with Rabin objectives [2]; and if so then we merge the whole EC as in line 9 of Algorithm 4; if not, and, moreover, there is no action out of the EC, we merge as in line 11 of Algorithm 4. This modified EC processing yields on-the-fly BRTDP and DQL algorithms for MDPs with Rabin objectives.

5 Experimental Results

Implementation. We have developed an implementation of our learning-based framework within the PRISM model checker [25], building upon its simulation engine for generating trajectories and explicit probabilistic model checking engine for storing visited states and U and L values. We focus on the complete-information case (i.e., BRTDP), for which we can perform a more meaningful comparison with PRISM. We implement Algorithms 1 and 2, and the on-the-fly EC detection algorithm of Sec. 4, with the optimisation of taking T as the set of all states explored so far.

We consider three distinct variants of the learning algorithm by modifying the GETSUCC function in Algorithm 1, which is the heuristic responsible for picking a successor state s' after choosing some action a in each state s of a trajectory. The first variant takes the unmodified GETSUCC, selecting s' at random according to the distribution $\Delta(s, a)$. This behaviour follows the one of the original RTDP algorithm [3]. The second uses the heuristic proposed for BRTDP in [31], selecting the successor $s' \in supp(\Delta(s, a))$ that maximises the difference $U(s') - L(s')$ between bounds for those states (M-D). For the third, we propose an alternative approach that systematically chooses all successors s' in a round-robin (R-R) fashion, and guarantees termination with certainty.

Results. We evaluated our implementation on four existing benchmark models, using a machine with a 2.8GHz Xeon processor and 32GB of RAM, running Fedora 14. We use three models from the PRISM benchmark suite [26]: *zeroconf*, *wlan*, and *firewire_impl_dl*; and a fourth one from [16]: *mer*. The first three use unbounded probabilistic reachability properties; the fourth a time-bounded probabilistic reachability. The latter is used to show differences between heuristics in the case of MDPs containing rare events, e.g., MDPs where failures occur with very low probability. All models, properties and logs are available online at [40].

We run BRTDP and compare its performance to PRISM. We terminate it when the bounds L and U differ by at most ε for the initial state of the MDP. We use $\varepsilon = 10^{-6}$ in all cases except *zeroconf*, where $\varepsilon = 10^{-8}$ is used since the actual values are very small. For PRISM, we use its fastest engine, which is the "sparse" engine, running value iteration. This is terminated when the values for all states in successive iterations differ

Name [param.s]	Param. values	Num. states	Time (s)				Visited states		
			PRISM	RTDP	M-D	R-R	RTDP	M-D	R-R
zeroconf $[N, K]$	$20, 10$	3,001,911	129.9	7.40	1.47	1.83	760	2007	2570
	$20, 14$	4,427,159	218.2	12.4	2.18	2.26	977	3728	3028
	$20, 18$	5,477,150	303.8	71.5	3.89	3.73	1411	5487	3704
wlan $[BOFF]$	4	345,000	7.35	0.53	0.48	0.54	2018	1377	1443
	5	1,295,218	22.3	0.55	0.45	0.54	2053	1349	1542
	6	5,007,548	82.9	0.50	0.43	0.49	1995	1313	1398
firewire_impl_dl $[delay, deadline]$	$36, 200$	6,719,773	63.8	2.85	2.62	2.26	26,508	28,474	22,038
	$36, 240$	13,366,666	145.4	8.37	7.69	6.72	25,214	26,680	20,219
	$36, 280$	19,213,802	245.4	9.29	7.90	7.39	32,214	28,463	25,565
mer $[N, q]$	$3000, 0.0001$	17,722,564	158.5	67.0	2.42	4.44	1950	3116	3729
	$3000, 0.9999$	17,722,564	157.7	10.9	2.82	6.80	2902	4643	4608
	$4500, 0.0001$	26,583,064	250.7	67.3	2.41	4.42	1950	3118	3729
	$4500, 0.9999$	26,583,064	246.6	10.9	2.84	6.79	2900	4644	4608

Table 1. Verification times using BRTDP (three different heuristics) and PRISM.

by at most ε. Strictly speaking, this is not guaranteed to produce an ε-optimal strategy (e.g. in the case of very slow numerical convergence), but on all these examples it does.

The experimental results are summarised in Table 1. For each model, we give the number of states in the full model, the time for PRISM (model construction, precomputation of zero/one states and value iteration) and time and number of visited states for BRTDP with each of the three heuristics described earlier. Some heuristics perform random exploration and therefore all results have been averaged over 20 runs.

We see that our method outperforms PRISM on all four benchmarks. The improvements in execution time on these benchmarks are possible because the algorithm is able to construct an ε-optimal policy whilst exploring only a portion of the state space. The number of states visited by the algorithm is at least two orders of magnitude smaller than the total size of the model (column 'Num. states'). These numbers do not vary greatly between heuristics.

The RTDP heuristic is generally the slowest of the three, and tends to be sensitive to the probabilities in the model. In the *mer* example, changing the parameter q can mean that some states, which are crucial for the convergence of the algorithm, are no longer visited due to low probabilities on incoming transitions. This results in a considerable slow-down, and is a potential problem for MDPs containing rare events. The M-D and R-R heuristics perform very similarly, despite being quite different (one is randomised, the other deterministic). Both perform consistently well on these examples.

6 Conclusions

We have presented a framework for verifying MDPs using learning algorithms. Building upon methods from the literature, we provide novel techniques to analyse unbounded probabilistic reachability properties of arbitrary MDPs, yielding either exact bounds, in the case of complete information, or PAC bounds, in the case of limited information. Given our general framework, one possible direction would be to explore other learning algorithms in the context of verification. Another direction of future work is to explore whether learning algorithms can be combined with symbolic methods for probabilistic verification.

Acknowledgement. We thank Arnd Hartmanns and anonymous reviewers for careful reading and valuable feedback.

References

1. Aljazzar, H., Leue, S.: Generation of counterexamples for model checking of Markov decision processes. In: QEST. pp. 197–206 (2009)
2. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)
3. Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. Artificial Intelligence 72(12), 81 – 138 (1995)
4. Bogdoll, J., Fioriti, L.M.F., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: FMOODS/FORTE. pp. 59–74 (2011)
5. Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and statistical model checking for modestly nondeterministic models. In: MMB/DFT. pp. 249–252 (2012)
6. Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: A flexible, distributable statistical model checking library. In: QEST. pp. 160–164 (2013)
7. Brázdil, T., Chatterjee, K., Chmelıik, M., Forejt, V., Křetínský, J., Kwiatkowska, M.Z., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. CoRR abs/1402.2967 (2014)
8. Bulychev, P.E., David, A., Larsen, K.G., Mikucionis, M., Poulsen, D.B., Legay, A., Wang, Z.: UPPAAL-SMC: Statistical model checking for priced timed automata. In: QAPL (2012)
9. Chatterjee, K., Henzinger, M.: An O(n^2) algorithm for alternating Büchi games. In: SODA. pp. 1386–1399 (2012)
10. Chatterjee, K., Henzinger, M.: Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In: SODA (2011)
11. Ciesinski, F., Baier, C., Grosser, M., Klein, J.: Reduction techniques for model checking Markov decision processes. In: QEST. pp. 45–54 (2008)
12. Courcoubetis, C., Yannakakis, M.: Markov decision processes and regular events (extended abstract). In: ICALP. pp. 336–349 (1990)
13. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Statistical model checking for networks of priced timed automata. In: FORMATS (2011)
14. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: CAV. pp. 349–355 (2011)
15. De Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis (1997)
16. Feng, L., Kwiatkowska, M., Parker, D.: Automated learning of probabilistic assumptions for compositional reasoning. In: FASE. pp. 2–17 (2011)
17. He, R., Jennings, P., Basu, S., Ghosh, A.P., Wu, H.: A bounded statistical approach for model checking of unbounded until properties. In: ASE. pp. 225–234 (2010)
18. Henriques, D., Martins, J., Zuliani, P., Platzer, A., Clarke, E.M.: Statistical model checking for Markov decision processes. In: QEST. pp. 84–93 (2012)
19. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI. pp. 307–329 (2004)
20. Jégourel, C., Legay, A., Sedwards, S.: Cross-entropy optimisation of importance sampling parameters for statistical model checking. In: CAV. pp. 327–342 (2012)
21. Jégourel, C., Legay, A., Sedwards, S.: A platform for high performance statistical model checking - PLASMA. In: TACAS. pp. 498–503 (2012)
22. Jégourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: CAV. pp. 576–591 (2013)
23. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer-Verlag (1976)
24. Kolobov, A., Mausam, Weld, D.S., Geffner, H.: Heuristic search for generalized stochastic shortest path mdps. In: ICAPS (2011)
25. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. pp. 585–591 (2011)

26. Kwiatkowska, M., Norman, G., Parker, D.: The PRISM benchmark suite. In: QEST. pp. 203–204 (2012)
27. Larsen, K.G.: Priced timed automata and statistical model checking. In: IFM (2013)
28. Lassaigne, R., Peyronnet, S.: Approximate planning and verification for large Markov decision processes. In: SAC. pp. 1314–1319 (2012)
29. Legay, A., Sedwards, S.: Lightweight Monte Carlo algorithm for Markov decision processes. CoRR abs/1310.3609 (2013)
30. Legay, A., Sedwards, S., Traonouez, L.: Scalable verification of markov decision processes. In: SEFM. pp. 350–362 (2014)
31. McMahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: ICML (2005)
32. Puterman, M.: Markov Decision Processes. Wiley (1994)
33. Rabih, D.E., Pekergin, N.: Statistical model checking using perfect simulation. In: ATVA. pp. 120–134 (2009)
34. Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: CAV. pp. 266–280 (2005)
35. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: CAV. pp. 202–215 (2004)
36. Strehl, A.L., Li, L., Wiewiora, E., Langford, J., Littman, M.L.: PAC model-free reinforcement learning. In: ICML. pp. 881–888 (2006)
37. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press (1998)
38. Younes, H., Simmons, R.: Probabilistic verification of discrete event systems using acceptance sampling. In: CAV. pp. 223–235 (2002)
39. Younes, H.L.S., Clarke, E.M., Zuliani, P.: Statistical verification of probabilistic properties with unbounded until. In: SBMF. pp. 144–160 (2010)
40. `http://www.prismmodelchecker.org/files/atva14learn/`