





The Modest State of Learning, Sampling, and Verifying Strategies

Arnd Hartmanns¹(✉)  and Michaela Klauk²(✉) 

¹ University of Twente, Enschede, The Netherlands

a.hartmanns@utwente.nl

² Saarland University, Saarbrücken, Germany

klauck@cs.uni-saarland.de

Abstract. Optimal decision-making under stochastic uncertainty is a core problem tackled in artificial intelligence/machine learning (AI), planning, and verification. Planning and AI methods aim to find good or optimal strategies to maximise rewards or the probability of reaching a goal. Verification approaches focus on calculating the probability or reward, obtaining the strategy as a side effect. In this paper, we connect three strands of work on obtaining strategies implemented in the context of the Modest Toolset: statistical model checking with either lightweight scheduler sampling or deep learning, and probabilistic model checking. We compare their different goals and abilities, and show newly extended experiments on Racetrack benchmarks that highlight the trade-offs between the methods. We conclude with an outlook on improving the existing approaches and on generalisations to continuous models, and emphasise the need for further tool development to integrate methods that find, evaluate, compare, and explain strategies.

1 Introduction

In many everyday interactions, but also in almost every system where software interfaces with and controls physical processes, decisions must be made in the presence of uncertainty about the possible actions' outcomes and the environment they interact with. In most cases, the uncertainty can be captured by randomisation: In casino card games, human players would like to maximise their chances of winning, or the expected return, against a randomly shuffled deck of cards. In travel planning, we have to choose between transport options that are often unreliable, and would like to maximise the probability of arriving on time. Our travel plans often include fallback options: the choices to make to recover when one step has gone wrong. Software controlling industrial processes

Authors are listed alphabetically. This work was supported by the German Research Foundation (DFG) under grant No. 389792660 as part of TRR 248, by the EU's Horizon 2020 research and innovation programme under MSCA grant agreement 101008233 (MISSION), and by NWO VENI grant no. 639.021.754.

must keep the process safe while optimising for, e.g., product completion time or throughput. Network routing over unreliable links must find routes that achieve a reasonable compromise between message delivery probability and expected delivery time. In autonomous driving, the car must react safely to unpredictable outside actors, imprecise measurements, and imperfect actuators without excessively inflating travel time. To appropriately deal with these kinds of situations, we need to find optimal, safe, or sufficiently performant *strategies*¹ describing the action to choose for every possible *state* of the system and/or environment.

The fundamental mathematical model for such scenarios are Markov decision processes (MDP). In MDP, the system jumps in discrete time steps from one discrete state to another. In every state, a *nondeterministic* choice (controllable or adversarial) over the available actions is followed by a *probabilistic* (i.e., random) choice of the next state. Various extensions cover continuous-time [16, 36, 79] and continuous-state [40, 42, 99] scenarios. The core problem, however, is always the same: *Find a strategy satisfying the stated objectives*. We focus on maximising the probability to eventually reach a set of goal states: *probabilistic reachability*.

Over the past decades, two broad types of solutions have been developed and implemented in tools. The *verification* approaches build on probabilistic and statistical model checking (PMC and SMC, respectively). PMC [8, 77] runs an iterative numeric algorithm on a representation of the full MDP to ϵ -approximate the maximal reachability probability. While the corresponding strategy can be extracted from the algorithm’s data structures upon termination, doing so has traditionally not been the focus of PMC. The strategy itself is typically represented as a list mapping (all reachable) states to chosen actions. SMC [1, 66, 80, 105] applies Monte Carlo simulation to a concise executable specification of an MDP—typically given in a higher-level modelling language such as Modest [16, 54] or JANI [22]—to estimate the probability under a given strategy. While highly effective in *evaluating* (the quality of) a strategy, it needs to be combined with a method to *find* an (optimal or good-enough) strategy in the first place. In contrast to PMC, SMC does not suffer from state space explosion: its memory usage is constant in the size of the MDP, as it only needs to store the current and next states obtained via the executable specification. It is thus a good partner for strategy-finding methods with similarly constant or moderate memory usage.

The first such method that we use in this paper is *lightweight scheduler sampling* (LSS) [81]: It randomly picks m strategies, applies an SMC-based heuristic to find the best one, and returns an SMC estimate under this strategy as an underapproximation of the maximum reachability probability. The key idea and advantage of LSS is its use of a constant-memory representation of a strategy as a fixed-size integer. It thus finds and evaluates a strategy in constant memory.

Finding good strategies is the focus of methods developed in (probabilistic) planning [26, 72, 104] and artificial intelligence/machine learning. A prominent success is *reinforcement learning* (RL) [100], in particular Q-learning [84, 85]. Here, again a concise specification of the MDP is executed in an initially random

¹ Depending on context, strategies are also called adversaries, policies, or schedulers.

manner, storing and over time improving a measure of the quality of every state-action pair that is executed. Similar to PMC, the strategy obtained by RL is a list mapping each *visited* state to the (best-quality) chosen action.

Deep neural networks (NNs) are responsible for astounding advances across applications as diverse as image classification [76], natural language processing [67], and playing games [98]. *Deep reinforcement learning* (deep RL) algorithms that use deep NNs to store the quality measure have exhibited unprecedented performance in various tasks [85]. At the cost of losing the eventual convergence to the optimum, deep RL reduces the memory usage of RL, possibly to constant (if we use a fixed-size NN independent of the size of the MDP or its executable specification). The combination of deep RL for finding strategies with SMC for their evaluation is *deep statistical model checking* (DSMC) [44].

So far, these methods have been presented, implemented, and benchmarked mostly in isolation. A wide range of PMC variants is available to users in tools such as PRISM [78], STORM [64], and the MODEST TOOLSET [57]. The latter, which is the focus of this paper, also includes the statistical model checker MODES [21] with support for LSS. The quantitative verification benchmark set (QVBS) [60] provides the standard set of models for benchmarking and comparisons of PMC and SMC tools. RL and in particular deep RL approaches, on the other hand, are often evaluated on training environments specified implicitly in the form of simulation code. In the academic context, the Arcade Learning Environment is widely used, which provides game simulators for different ATARI 2006 benchmarks [12]. For reinforcement learning, these training environments are then often interfaced with the learning algorithm via the OpenAI Gym API [19]. It is used by algorithms interacting with the interface [35, 47, 68, 89, 97], as well as benchmarks that implement (and sometimes extend) it [10, 27, 37, 102, 103, 106]. We use the OpenAI Gym API via MOGYM [45] to train NN and then evaluate their quality with DSMC.

This paper contributes a uniform presentation of PMC, SMC with LSS, and deep RL with DSMC, spanning the range from verification approaches delivering optimal strategies at the cost of state space explosion to AI-based methods using deep NN approximations for limiting memory usage at the cost of losing optimality guarantees. We spell out the consequences that the differing goals of these methods have for obtaining and ultimately explaining strategies. Our new experimental comparison in Sect. 4 confirms the expected differences, but also highlights the particularities and similarities of the approaches. For example, the effectiveness of LSS appears to be an indicator for the (startup) difficulty of RL on our models. We use different variants of Racetrack benchmarks embodying a simplistic autonomous driving scenario. They are easy to visualise and understand, yet can flexibly be configured to provide various kinds of difficulties for the methods we study. Their action space is also very regular, which is currently a prerequisite for deep RL to be effective.

2 Preliminaries

For any nonempty set S we let $\mathcal{D}(S)$ denote the set of discrete probability distributions over S , i.e., of functions $\mu: S \rightarrow [0, 1]$ such that the support $\text{spt}(\mu) = \{s \in S \mid \mu(s) > 0\}$ is countable and $\sum_{s \in \text{spt}(\mu)} \mu(s) = 1$.

Definition 1. A finite Markov decision process (MDP) [13, 69, 92] is a tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, s_0, \mathcal{S}_* \rangle$$

consisting of a finite set of states \mathcal{S} , a finite set of actions \mathcal{A} , the partial transition probability function $\mathcal{T}: \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{D}(\mathcal{S})$, a reward function $\mathcal{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}_{\geq 0}$ assigning a reward to each triple of state, action, and target state, an initial state $s_0 \in \mathcal{S}$, and a set of goal states $\mathcal{S}_* \subseteq \mathcal{S}$. For every state $s \in \mathcal{S}$, there is at least one action $a \in \mathcal{A}$ such that $\mathcal{T}(s, a)$ is defined.

An action $a \in \mathcal{A}$ is *applicable* in a state $s \in \mathcal{S}$ if $\mathcal{T}(s, a)$ is defined. In this case, we also write $\mathcal{T}(s, a, t)$ for the probability $\mu(t)$ of going to state t according to $\mathcal{T}(s, a) = \mu$. $\mathcal{A}(s) \subseteq \mathcal{A}$ is the set of all actions that are applicable in state s . An infinite sequence of states connected via transitions with applicable actions, $\zeta = (s_i)_{i \in \mathbb{N}}$, is a *path*.

The reward function assigns to every transition from one state to another a reward depending on the start and destination state as well as the action. This enables us, e.g., to reason about the sum of the rewards obtained when taking multiple transitions in a row. Rewards, while not influencing reachability probabilities, are an important concept in RL that we come back to in Sect. 3.3.

Definition 2. Given an MDP \mathcal{M} as above, a function

$$\sigma: \mathcal{S} \rightarrow \mathcal{A}$$

satisfying $\sigma(s) \in \mathcal{A}(s)$ for all states s is a (*deterministic*) memoryless strategy.

A strategy determines the action to take for every state. Restricting MDP \mathcal{M} to the choices made by strategy σ results in an *induced* discrete-time Markov chain (DTMC) $M|_\sigma$: an MDP where $\forall s \in \mathcal{S}: |\mathcal{A}(s)| = 1$. Intuitively, the probability that a certain path (or prefix of it) is taken in a DTMC can be calculated as the product over the transition probabilities of the path (prefix). Formally, since the set of paths is uncountable, the cylinder set construction [77] can be used to obtain a probability measure P over paths such that in particular the set $\Pi_{\mathcal{S}_*}$ of paths that contain a state in \mathcal{S}_* is measurable. Then $P(\Pi_{\mathcal{S}_*})$ is the *reachability probability* $p_{\mathcal{S}_*}$ in the DTMC. In an MDP, each strategy σ induces a DTMC $M|_\sigma$ and consequently a reachability probability $p_{\mathcal{S}_*}^\sigma$. Then $p_{max} \stackrel{\text{def}}{=} \sup_\sigma p_{\mathcal{S}_*}^\sigma$ is the maximum reachability probability that we are looking for; and an *optimal strategy* σ_{max} such that $p_{max} = p_{\mathcal{S}_*}^{\sigma_{max}}$ in fact exists [14].

3 Finding Strategies

To find strategies that satisfy stated criteria, e.g., (near-)optimality or suitability for a certain purpose, various approaches have been developed in the fields of probabilistic verification and artificial intelligence (AI). In this section, we contrast (i) the traditional verification approach of probabilistic model checking, (ii) the more recent lightweight scheduler sampling method that lifts statistical model checking from DTMC to MDP, (iii) the core AI technique of reinforcement learning, and (iv) its variant using deep neural networks to approximate the Q-function combined with statistical model checking.

All approaches start with a succinct executable specification of an MDP. This is either a model specified in a (textual or graphical) modelling language, for which execution support is provided by some tool's state space exploration engine, or a computer program directly implementing the model. For simplicity, we only assume that we have an interface with the following functions:

- `initial()` to obtain s_0 ,
- `actions(s)` to obtain $\mathcal{A}(s)$,
- `sample(s, a)` to (pseudo-)randomly select a next state s' according to $\mathcal{T}(s, a)$,
- `distr(s, a)` to obtain the distribution $\mu = \mathcal{T}(s, a)$, e.g., as a list of pairs of probabilities p and next states s' such that $p = \mu(s')$, and
- `goal(s)` that returns *true* if $s \in \mathcal{S}_*$ and *false* otherwise.

Not all functions will be needed by all approaches; e.g., probabilistic model checking uses `distr` but not `sample` whereas reinforcement learning does the opposite.

3.1 Probabilistic Model Checking

Traditional exhaustive probabilistic model checking (PMC) starts by constructing a complete in-memory representation of the MDP: from a call to `initial`, it performs a graph search by iteratively following all transitions via calls to `actions` and `distr` until no new states are discovered. The resulting MDP as in Definition 1 can be stored as an *explicit-state* graph-like data structure or sparse matrix, or *symbolically* using multi-terminal binary decision diagrams [88].

The next step in PMC is to calculate p_{max} . One approach is to convert the MDP into a linear program, with one variable per state, which in turn is solved using any linear programming solver. The value of the variable corresponding to the initial state will then be p_{max} . Although this approach can in principle deliver exact results (though usually up to some floating-point precision), it needs to store the MDP in memory twice (as in Definition 1 and in suitably-encoded form for the LP solver), and most LP solvers so far do not scale well to large MDP. Therefore, most PMC tools default to using iterative numeric algorithms based on value iteration. Value iteration computes a sequence of values $v_i(s)$ for every state $s \in \mathcal{S}$ that converges to the maximum reachability probability *from each state* (i.e. as if that state was the initial state). It does so by, starting from the

trivial underapproximation where $\forall i: v_i(s) = 1$ if $s \in \mathcal{S}_*$ and $v_0(s) = 0$ if $s \notin \mathcal{S}_*$, applying the Bellman equation

$$v_{i+1}(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \text{spt}(\mathcal{T}(s,a))} \mathcal{T}(s, a, s') \cdot v_i(s)$$

for all states $s \notin \mathcal{S}_*$. Then $\lim_{i \rightarrow \infty} v_i(s_0) = p_{max}$. Value iteration lacks a stopping criterion to determine the i where $v_i(s_0)$ is close enough to the true value (e.g., within a user-specified relative error ϵ) [48]. Thus sound PMC tools today use variants of value iteration that provide such a stopping criterion, e.g., interval iteration [18, 49], sound value iteration [93], or optimistic value iteration [59].

Strategy Representation. Once iterations stop at some i , a (near-)optimal strategy has implicitly been obtained, too: it is, assuming no end components here,

$$\sigma_{max} = \{ s \mapsto \arg \max_{a \in \mathcal{A}(s)} \sum_{s' \in \text{spt}(\mathcal{T}(s,a))} \mathcal{T}(s, a, s') \cdot v_i(s) \}.$$

The PMC tools PRISM, STORM, and MCSTA of the MODEST TOOLSET all offer an option to not only report p_{max} , but also write σ_{max} to file as a list of state-action pairs, with as many entries as there are states in the MDP. With typical ϵ -correct relative-error implementations of sound value iteration variants, PMC tools guarantee that, when stopped,

$$|v_i(s_0) - p_{max}| / p_{max} \leq \epsilon$$

(assuming non-zero p_{max} ; probability-zero states can be determined by graph-based precomputations [38]). The value of ϵ is specified by the user, and typically 10^{-3} or 10^{-6} by default.

Modest Tools. In the MODEST TOOLSET, the MCSTA tool implements PMC. It is an explicit-state model checker that can use secondary storage (i.e., hard disks and SSDs) to mitigate state space explosion to some degree at the cost of runtime [58]. Its focus is on providing correct results; for this purpose, it implements interval iteration, sound value iteration, and optimistic value iteration, and recently gained the ability to obtain results that are guaranteed to be free of errors due to imprecisions and rounding in floating-point calculations [56]. It was the first tool to implement practically efficient methods for reward-bounded properties [50], includes a novel symblic engine to handle very large structured models [51], and provides methods that work with only a partial exploration of the state space [4, 18]. Beyond MDP, it has state-of-the-art support for Markov automata [24] and stochastic timed automata [53]. The QComp 2020 and 2021 competitions [23, 52] showed that MCSTA performs well.

Related Methods. Other alternatives to linear programming and value iteration are policy iteration [70] and variants such as topological value iteration [28] that may deliver significant speedups for MDPs with appropriately-sized strongly connected components. To mitigate state space explosion, we can attempt to only explore a part of the state space that is likely to be reached [18]; then we

obtain an upper bound on the reachability probability by assuming all unexplored “border states” to be goal states, and a lower bound by assuming them to be non-goal states. Often referred to as (being based on) BRTDP [83], an approach from probabilistic planning, implementations nowadays—such as the one in MCSTA—are rather different from the original BRTDP technique and better described as partial exploration-based PMC [4]. A similar approach also known from probabilistic planning called LRTDP [17] in combination with FRET [75] has recently been extended to be applicable to all established property types, except long-run averages and nested properties, on MDP structures with positive and zero-valued rewards [73]. With this technique, often only a fraction of the state space—the part sufficient to calculate the property at hand—has to be visited. The technique is implemented in MODYSH in the MODEST TOOLSET.

3.2 Statistical Model Checking with Scheduler Sampling

Given our interface to an MDP and a function `schedule(s)` implementing a strategy σ to return $\sigma(s)$, statistical model checking (SMC) estimates the reachability probability on the DTMC induced by σ up to a statistical error. It does so by sampling the indicator function on paths

$$\mathbb{1}_{\Pi_{\mathcal{S}_*}} \stackrel{\text{def}}{=} \{ \pi \mapsto 1 \text{ if } \pi \text{ contains a state in } \mathcal{S}_* \text{ else } 0 \}$$

n times as follows:

1. Initialise $s := \text{init}()$.
2. If `goal(s)`, return 1; if the probability to reach a goal state from s is 0, return 0.
3. Select an action $a := \text{schedule}(s)$ and sample the next state: $s := \text{sample}(s, a)$.
4. Go to step 2.

That is, SMC generates n *simulation runs*. Let k be the number of runs where 1 is returned; then the sample mean $\hat{p}_{\mathcal{S}_*} \stackrel{\text{def}}{=} k/n$ is an unbiased estimator for $p_{\mathcal{S}_*}^{\sigma}$. This basic approach can be modified in various ways to incorporate different statistical methods that quantify the error to be expected from an SMC analysis. Popular methods are to compute confidence intervals given n and either a desired interval width or confidence level; to perform sequential testing using Wald’s sequential probability ratio test [101], thereby dynamically determining n as the samples come in; or to use the Okamoto bound [87] that provides a formula relating n , the confidence level δ , and the error ϵ a priori such that

$$\mathbb{P}(|\hat{p}_{\mathcal{S}_*} - p_{\mathcal{S}_*}| > \epsilon) < 1 - \delta$$

with typical default values of δ being 0.95 or 0.99. For a more extensive overview of statistical methods for SMC, we refer the interested reader to the survey by Reijsbergen et al. on hypothesis testing and its references [94]. The development of efficient methods to determine whether a state has probability 0 of reaching

a goal state in step 2 is a topic of ongoing research [5]. When $p_{\mathcal{S}_*}$ is small, ϵ must also be small for the result to be useful. Then the n required to achieve the same confidence grows quickly, leading to a runtime explosion. This is the *rare events* problem faced by SMC, for which various *rare event simulation* [95] methods exist as mitigation.

As specified above, SMC requires a strategy to be given in order to be applicable to MDP. Many SMC tools do not provide support for user-specified strategies, but instead implicitly use the uniform random strategy that, every time `schedule(s)` is called, uniformly at (pseudo-)random samples a new action from $\mathcal{A}(s)$. The result consequently is *some* probability *somewhere* between maximum and minimum. UPPAAL SMC [34] notably defines a “stochastic semantics” for probabilistic timed automata that makes continuously uniformly- or exponentially-distributed choices over time delays followed by discretely uniform choices over actions. The resulting non-nondeterministic model is sometimes referred to as stochastic timed automata, not to be confused with the earlier formalism of the same name of [16] that is a proper extension of probabilistic timed automata preserving their nondeterminism.

Lightweight scheduler sampling identifies a strategy by a fixed-size (typically 32-bit) integer value. It

1. (pseudo-)randomly selects m such *strategy identifiers*, then
2. applies a heuristic involving SMC runs as described above under the sampled strategy to try to find the one that induces the highest probability, and finally
3. performs another SMC analysis for the selected strategy (statistically independent from step 2) to obtain an estimate of the probability it induces.

The result is an underapproximation of p_{max} , up to statistical errors. LSS may or may not find a strategy better than the uniform random one, but often does so. Its ability to find a near-optimal strategy depends only on the probability mass of near-optimal strategies in the space of strategies sampled from.

The key idea that makes LSS work, in constant memory in the size of the MDP, lies in its implementation of `schedule`. It uses a hash function \mathcal{H} that takes an arbitrary-length bitstring and returns a 32-bit integer such that, ideally, (i) small changes in the input result in unpredictable and significant changes in the output, and (ii) for uniformly random inputs (e.g., of a fixed length), the outputs appear uniformly distributed over the output space. Then it implements `schedule(s)` for strategy identifier $\mathfrak{s} \in \mathbb{Z}_{32}$ by selecting the $(\mathcal{H}(s.\mathfrak{s}) \bmod |\mathcal{A}(s)|)$ -th element of a fixed ordering of $\mathcal{A}(s)$, where $s.\mathfrak{s}$ is the concatenation of the bitstring representations of s and \mathfrak{s} . In this way, `schedule` implements a memoryless strategy as required and sufficient for unbounded probabilistic reachability, but also other types of strategies—such as history-dependent or partial-information strategies [29]—are easy to implement by appropriately changing the input of \mathcal{H} .

We use LSS with a simplified variant of the *smart sampling* [32] heuristics for step 2. In addition to m , it is parametrised by $n_r \geq m$, the simulation budget per round. In the first round, we perform $\lfloor n_r/m \rfloor$ runs for each of the m strategy identifiers. Usually, n_r is not much larger than m , so this first round produces a very coarse estimation of the quality of each sampled strategy. We then drop the

worst-performing half of the strategies before proceeding with the second round, where $\lfloor n_r/2m \rfloor$ runs are performed per strategy, providing a somewhat better estimation. This process, dropping the worst half of the remaining strategies in every round, continues until only one strategy remains. In this way, we can evaluate a large number of strategies with moderate simulation effort.

Strategy Representation. SMC with LSS returns the estimate of the reachability probability under the best strategy, and the 32-bit integer identifying that strategy. By itself, this integer is useless: it does not describe the strategy’s decisions directly. However, given a state of the MDP and the known implementation of `schedule` used during the LSS process, we can recompute the strategy’s decision at any time. We thus get a memory-efficient strategy representation that is not explanatory in any way.

Modest Tools. In the MODEST TOOLSET, the MODES statistical model checker [21] implements SMC with LSS as described above. It has dedicated simulation algorithms for MDP, Markov automata, singular stochastic hybrid automata, and general stochastic hybrid automata: as the modelling formalism becomes more expressive, simulation becomes computationally more involved, with more and more complex computations needed for every transition (up to numeric integration to approximate the non-linear dynamics in general stochastic hybrid automata). To mitigate the rare events problem, MODES implements rare event simulation by means of importance splitting in a highly automated fashion [20].

Related Methods. Prior to implementing LSS, MODES used partial order [15] and confluence reduction [61] checks to identify whether the nondeterministic choices in an MDP it simulates are non-spurious, i.e., whether they influence the probability being estimated. If not, these choices would be resolved randomly; otherwise, simulation would abort with an error indicating the presence of possibly non-spurious nondeterminism. Where UPPAAL SMC implicitly applies a specific strategy—its stochastic semantics—the more recent UPPAAL Stratego tool [33] combines SMC with the computation of (most permissive) strategies.

3.3 Reinforcement Learning

Reinforcement learning is an AI approach to train agents to take actions maximising a reward in uncertain environments. Mathematically, the agent in its environment can be described as an MDP: the agent chooses actions; the environment determines the states and is responsible for the probabilistic outcomes of the actions. In this paper, we follow the *Q-learning* approach: We maintain a Q-function $Q: \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ initialised arbitrarily, or to 0 everywhere. Using a learning rate parameter α , a discounting factor $\gamma \in (0, 1]$, and a probability ϵ that is initially 1, n learning *episodes* are performed as follows:

1. Set $s := \text{init}()$.
2. Perform option a) with probability ϵ and b) with probability $1 - \epsilon$:
 - a) Select a from $\text{actions}(s)$ uniformly at random (exploration).

- b) Select $a := \arg \max_{a' \in \text{actions}(s)} Q(s, a')$ (exploitation).
3. Sample $s' := \text{sample}(s, a)$ and set $r := 1$ if $\text{goal}(s')$ and 0 otherwise.
4. Update $Q(s, a) := Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a' \in \text{actions}(s')} Q(s', a') - Q(s, a))$.
5. Set $s := s'$; if $\text{goal}(s)$ or s has probability 0 of reaching a goal state, end the episode; else go to step 2.

An episode is very similar to a simulation run, except that we update the Q-function to estimate the “quality” of taking action a from state s as $Q(s, a)$ and follow an “ ϵ -greedy” strategy: initially, when $\epsilon = 1$, we explore randomly; over time, we make it more and more likely to follow what looks like the best action to improve our estimate of its quality. RL traditionally optimises for expected discounted rewards, thus the discounting factor γ ; for unbounded reachability probabilities, we set γ to 1 and only obtain a reward upon reaching a goal state (as above). As long as we are guaranteed to visit every state infinitely often, $\max_{a \in \mathcal{A}(s_0)} Q(s_0, a)$ converges towards p_{max} . Like in value iteration, there is no stopping criterion that would ensure a specified error.

Strategy Representation. The (near-)optimal strategy obtained when we end the learning process is directly given by the Q-function: It is

$$\sigma_{max} = \{ s \mapsto \arg \max_{a \in \mathcal{A}(s)} Q(s, a) \}.$$

Like in PMC, if we have an explicit in-memory representation of the Q-function, we can write this strategy to file as a list of state-action pairs.

Modest Tools. RL with an explicit representation of the Q-function is implemented in the MODEST TOOLSET’s MODES tool to find strategies in non-linear stochastic hybrid automata, where classic PMC techniques cannot be applied due to the continuous nature of the state space [86].

Related Methods. The first use of RL for formal models known to us is in the work of Henriques et al. [63], which however neglects the statistical error incurred by performing repeated tests. The first sound formal use of RL is in [18]. For probabilistic reachability, the rewards are very sparse: only when we hit a goal state we do receive a reward. This tends to make RL inefficient; for linear-time properties, denser reward structures can automatically be created, see, e.g., [55].

3.4 Deep Statistical Model Checking

Deep Reinforcement Learning. In RL as described above, we need to store the Q-function in memory; as we visit more states in large MDP, this will lead to state space explosion as in PMC. To avoid this scalability limitation, the use of function approximators to store an inexact representation of the Q-function has become popular in recent years. In particular, when we use deep neural networks as a function approximator with RL, we perform *deep reinforcement learning* resp. deep Q-learning [39, 85]. This use of artificial neural networks

(NN) to learn strategies in large systems has seen dramatic successes, exhibited by the abilities of today’s AI systems to play, win, and solve games such as Atari games [85], Go and Chess [98], and Rubic’s cubes [2].

NN consist of neurons: atomic computational units that apply a (non-linear) *activation function* to a weighted sum of their inputs [96]. We consider feed-forward NN, where neurons are arranged in a sequence of layers. Inputs are provided to the first (input) layer, and the computation results are propagated through the layers in sequence until reaching the final (output) layer. In every layer, every neuron receives as inputs the outputs of all neurons in the previous layer. For a given set of possible inputs \mathcal{I} and (final layer) outputs \mathcal{O} , a neural network can be considered as an efficient-to-query total function $\pi: \mathcal{I} \rightarrow \mathcal{O}$. For the problems discussed in this paper, one can assume that this function constitutes the strategy σ : the inputs are the states \mathcal{S} and the outputs are actions from \mathcal{A} . *Deep* neural networks consist of many layers.

Strategy Representation. A NN represents a Q-function, and thus a strategy. The NN used in deep RL are typically initialised with random weights, representing a random initial Q-function. As we learn more and more episodes, the decisions determined as optimal by the NN tend to converge towards a good, often optimal, strategy. However, in contrast to the exact Q-learning of Sect. 3.3, deep RL does not *need to* converge in this way, and in practice rarely behaves monotonically. That is, more episodes can (temporarily) make the NN represent a worse strategy. To preserve the memory advantage of NN over explicit representations, at the end of the learning process, we store the NN itself (i.e., its structure and weights) as the strategy instead of turning it into a list of state-action pairs (which would require a full state space exploration). The disadvantage of this representation is that, similar to the scheduler identifiers of LSS, the NN definition itself neither makes the strategy’s decisions explicit nor explains them.

Deep Statistical Model Checking. In contrast to Q-learning with an exact representation of the Q-function, we cannot rely on the value returned by the NN for the initial state being in any formal way related to p_{max} [44]. One approach to assess the quality of strategies given by NN is *deep statistical model checking* (DSMC) [44], which bridges machine learning and verification: first, deep RL delivers a strategy σ in the form of an NN trained to act and achieve a certain goal in an environment described by a formal model. Second, SMC, as a verification technique, assesses the quality of the strategy defined by the NN. This is done by implementing the `schedule` function in SMC as described in Sect. 3.2 by querying the NN as a black-box oracle: The NN receives the state descriptor s as input, and it returns as output a decision $\sigma(s)$ determining the next step. Hence, at the core of DSMC is a straightforward variation of SMC, applied to an MDP, together with an NN that has to take the decisions. The DSMC approach furthermore allows assessing the progress of the NN during learning. As shown in works on DSMC [44, 46], the quality assessment of an agent during training is not trivial and cannot always be derived from the observed training returns.

Modest Tools. The DSMC functionality of using a previously trained NN to resolve the nondeterminism during SMC is implemented in a branch of MODES [44] that will be integrated into the official version of the MODEST TOOLSET soon. In addition, this DSMC extension of MODES is used in MOGYM [45]. MOGYM is a toolbox that bridges the gap between formal methods and RL by enabling (a) formally specified training environments to be used with machine-learned decision-making agents, and (b) the rigorous assessment of the quality of learned agents. For (a), it implements and extends the OpenAI Gym API [19]. MOGYM is based on Momba [74], a Python toolbox for dealing with quantitative models from construction to analysis centred around JANI. MOGYM can process JANI models for the description of a training environment and, based on the induced formal MDP semantics, makes it possible to train agents using popular RL algorithms. For (b), the environment format itself is accessible to state-of-the-art model checkers. This enables to perform DSMC by using MODES directly in MOGYM. The DSMC extension of MODES is also integrated in *DSMC evaluation stages* [46], where DSMC is applied during deep RL to determine state space regions with weak performance to concentrate on during the learning process. To visualise the SMC results of MODES when executing DSMC on Racetrack benchmarks, the tool *TraceVis* has been implemented [43]. It takes the traces generated by MODES as input, visualises and clusters them, and provides information on the goal probability when starting on a predefined position.

Related Methods. Other works combining formal methods with NN, for example, study strategy synthesis for partially observable MDPs (POMDPs) using recurrent neural networks (RNN). The RNN is then used to construct a Markov chain for which the temporal property can be checked using PMC [25]. Furthermore, an iterative learning procedure consisting of SMT-solving and learning phases has been used to construct controllers for stochastic and partially unknown environments [71]. In addition, a reinforcement learning algorithm has been invented to synthesize policies which fulfil a given linear-time property on an MDP [62]. By expressing the property as a limit deterministic Büchi Automaton, a reward function over the state-action pairs of the MDP can be defined such that the policy is only constructed by considering the part of the MDP which fulfils the property. This is of special interest when working on sparse reward models. To be able to add features to NN acting as a controller without retraining and losing too much performance, quantitative run-time shields have been devised [7]. This method can easily be implemented as an extension of DSMC. The shields may alter the command given by the controller before passing it to the system under control. To generate these shields, reactive synthesis is used.

3.5 Summary

The four approaches we presented provide distinct characteristics and advantages as well as drawbacks: PMC delivers precise results up to a user-specified error ϵ , and RL eventually converges to the true result as well with statistical

guarantees [18], whereas SMC with LSS and deep RL in DSMC cannot be guaranteed to eventually obtain a near-optimal strategy. In all but PMC (using sound algorithms such as interval iteration), there is no unconditional stopping criterion to determine when ϵ is reached. PMC is thus the only technique that can guarantee optimality. This comes at the cost of memory usage: the state space explosion problem. RL faces the same issue, where it however can be avoided by using NN trained with deep RL in DSMC—at the cost of the eventual convergence guarantee. Where PMC needs the `distr` method of our MDP interface, `sample` suffices for the others. That is, PMC requires a white-box model, whereas the other methods only need sampling access—a much simpler requirement for practical applications. Rare events are no issue for PMC, but lead to a runtime explosion in SMC, and similarly hinder RL and DSMC, where the learning process will be very unlikely to ever explore a path leading to one of the rare goal states. Finally, in terms of the representation of the strategy, PMC and RL deliver explicit and complete strategies, which however may be unmanageably large, whereas LSS and DSMC provide compact yet opaque representations.

4 Experiments

We compare the approaches presented above on a set of six Racetrack benchmarks differing in the track shape. Originally Racetrack is a pen and paper game [41]: A track is drawn with a start line and a goal line on a sheet of squared paper. A vehicle starts with velocity 0 from some position on the start line, with the objective to reach the goal as fast as possible without crashing into a wall. Nine possible actions modify the current velocity vector by one unit (up, down, left, right, four diagonals, keep current velocity). This simple game lends itself naturally as a benchmark for sequential decision making. Like Barto et al. [11], we consider a noisy version of Racetrack that emulates slippery road conditions: actions may *fail* with a given probability, in which case the action does not change the velocity and the vehicle instead continues driving with unchanged velocity vector. In particular, when extending the problem with noise, we obtain MDP that do not necessarily allow the vehicle to reach the goal with certainty. In a variety of such noisy forms, Racetrack was adopted as a benchmark for MDP algorithms in the AI community [11, 17, 82, 90, 91]. Because of its analogy to autonomous driving, Racetrack has recently also been used in multiple verification and model checking contexts [9]. Due to the velocity vector only taking integer values, Racetrack benchmarks are discrete-state models; by definition, they are discrete-time.

Experimental Setup. We performed our experiments on Racetrack benchmarks with a noise probability of 10%. For each Racetrack instance, given as a JANI model, we use PMC, SMC with LSS, and deep RL with DSMC to find a good or optimal strategy for reaching the goal line from a certain start position and compute its induced probability. For PMC, we used MCSTA on an Intel Core i7-6600U system (2 cores, 4 threads) with 16 GB of RAM running 64-bit Windows 10. For SMC with LSS, we used MODES on an Intel Core i7-4790 system

Table 1. *barto-small* results, unbounded.

Method	p	Time	Episodes
PMC	1.000	362	–
DSMC	0.000	600	4,000
	0.981	5,280	44,000

Table 2. Results for *maze*.

Method	p	Time	Episodes
PMC	0.968	1,305	–
DSMC	0.000	600	14,000
	0.000	55,800	981,000
SMC-LSS	0.000	684	–

property. This model has only 44,231,073 states, which MCSTA easily handles in 16 GB of memory, and thus finds the optimal result (and strategy) with a probability close to 1 in 6 min. In deep RL with DSMC, on the other hand, the NN has not learnt any useful strategy at this point, after 4,000 learning episodes. After 88 min, however, it has found action choices that result in a near-optimal probability of 0.981. In Table 3, we see how the uninformed sampling employed by LSS performs in comparison. We underline the best results found for each value of m . Using a population of $m = 100,000$ strategies, LSS already finds some that lead the vehicle to the goal, albeit with a probability of at most 0.281. Once we extend the population to 1,000,000 strategies, the success probability increases to about 0.35. We observe limited returns in sampling larger numbers of strategies: with $m = 10,000,000$ for family 3, we get a probability of approx. 0.426 after 192 min—another increase of around 0.07 for a tenfold increase of m (and $20\times$ of runtime). So in runtime comparable to PMC, LSS finds non-trivial strategies, but unlike for deep RL with DSMC, additional time does not lead to significant further improvements. Our use of different step bounds highlights a peculiar effect here: although strategies exist that reach the goal with a high probability in 100 steps (as found by PMC), LSS fails to find these; the reasonable strategies it finds need at least 200 steps, but do not improve when given more steps to reach the goal. As a baseline, the uniform random strategy is clearly useless, essentially never allowing the vehicle to reach the goal.

Table 3. Results for *barto-small*, step-bounded analysis.

Method	m	fam.	100 steps		200 steps		400 steps		800 steps		
			p	Time	p	Time	p	Time	p	Time	
PMC			0.913	172	1.000	231	1.000	335	1.000	335	
SMC-LSS	10,000	1	0.000	2	0.004	3	0.004	4	0.004	5	
		2	0.000	2	0.000	3	0.000	3	0.000	5	
		3	0.000	3	<u>0.028</u>	3	<u>0.027</u>	4	<u>0.032</u>	5	
	100,000	1	0.000	24	0.041	35	0.044	43	0.043	58	
		2	0.000	23	<u>0.281</u>	39	<u>0.281</u>	48	<u>0.281</u>	62	
		3	0.000	26	0.225	39	0.225	47	0.225	58	
	1,000,000	1	0.286	264	0.343	396	0.343	476	0.343	630	
		2	0.318	250	0.317	398	0.317	481	0.317	620	
		3	<u>0.351</u>	258	<u>0.351</u>	394	<u>0.350</u>	470	<u>0.349</u>	601	
		10,000,000	3						<u>0.426</u>	11,513	
	SMC/unif.			0.000	0	0.000	0	0.000	0	0.000	0

4.2 The *barto-medium* and *barto-big* Tracks

We next consider the three variants of an R-shaped track shown in Fig. 2: the original *barto-big* track on the right, and two versions of reduced size. We use these variants of presumably increasing difficulty to find the point where our three methods stop being able to find non-trivial strategies. The MDP for track *barto-medium-small* has 35,149,687 states, *barto-medium-large* has 69,918,581, and *barto-big* has 330,872,263. The experimental results are shown in Table 4. For PMC, MCSTA manages to check the two *barto-medium* benchmarks without running out of memory, but on *barto-big*, 16 GB of memory do not suffice. In its “hybrid” disk-based mode, where the MDP and all value vectors are kept in memory-mapped files, MCSTA manages even this largest model, at a significant runtime cost. Deep RL for DSMC again does not manage to learn a useful NN in up to 30 min for all sizes of the *barto* tracks but is able to deliver a near-optimal strategy after 741,000 episodes for *barto-medium-small*. The slightly increased difficulty of *barto-medium-large* is already enough such that the NN has still a poor performance with a goal probability of 0.002 after 641,000 episodes which took 18 h and 40 min. *barto-big* is then too difficult to learn a strategy reaching the goal even after 29 h with 665,000 training episodes when using the normal start setting. From other works we know finding a good strategy is no issue for deep RL in the random start setting [44, 46]. For LSS, we only show the best result achieved among the three families with $m = 1,000,000$ and a bound of 800 steps. The *barto-medium-small* track is the last one where LSS manages to find non-trivial strategies, however these strategies already perform very poorly. Our

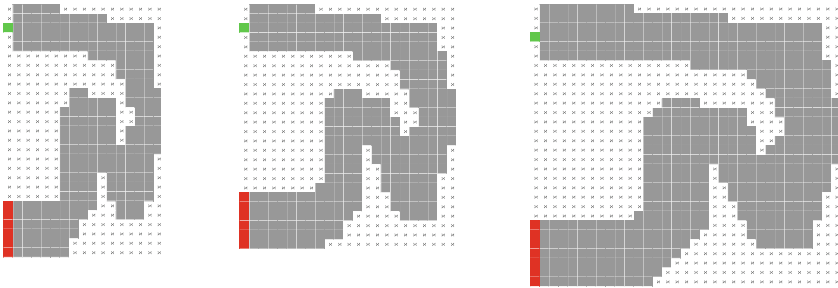


Fig. 2. The *barto-medium-small*, *barto-medium-large*, and *barto-big* tracks.

Table 4. Results for the two *barto-medium* tracks and the *barto-big* track.

Method	<i>barto-medium-small</i>			<i>barto-medium-large</i>			<i>barto-big</i>		
	p	Time	Episodes	p	Time	Episodes	p	Time	Episodes
PMC	0.999	156	–	1.000	402	–	1.000	17,413	–
DSMC	0.000	600	8,000	0.000	600	8,000	0.000	600	5,000
	0.000	1,800	21,000	0.000	1,800	18,000	0.000	1,800	11,000
	0.946	50,400	741,000	0.002	67,200	641,000	0.000	104,400	665,000
SMC-LSS	0.023	670	–	0.000	742	–	0.000	1,048	–

explanation for this result, despite the *barto-medium* models having a similar number of states as *barto-small*, is that the tracks require much more specific behaviour to navigate the R shape, i.e., fewer strategies are successful and thus the probability mass of successful strategies becomes too low for LSS. In essence, LSS hits a “rare strategy problem”. We do not show the uniform random strategy, which again does not manage to hit the goal; in fact, it does not manage to do so for any of our examples in this section.

4.3 The *maze* Track

The *maze* track is depicted in Fig. 3. The MDP for this track consists of 156,967,073 states. The results of the experiments are summarized in Table 2. MCSTA implementing PMC solves the benchmark in around 22 min. While we do not need to use its hybrid disk-based mode, it still needs more than 16 GB of memory at certain points and is thus slowed down by the operating system swapping memory contents to disk. Because of the very narrow streets on the track, LSS has no chance: any useful strategy needs to pick a long sequence of exactly the right actions to not crash into a wall, which is very unlikely to be sampled. Consequently, we only see probability-zero strategies in our experiments. The same issue makes it infeasible to learn an NN of reaching the goal with deep RL in the normal start setting: the random exploration phase most likely never manages to hit any goal state and thus obtain a positive reward. Even after 15.5 h and 981,000 training episodes, we found

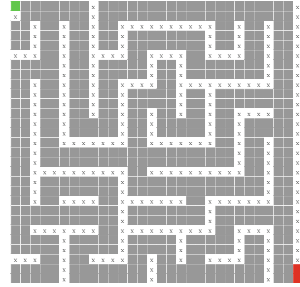


Fig. 3. The *maze* track.

Even after 15.5 h and 981,000 training episodes, we found

Table 5. Results for *river-deadend-narrow*.

Method	m	fam.	p	Time	Episodes
PMC	–	–	1.000	1,563	–
DSMC	–	–	0.000	1,800	18,000
	–	–	0.984	36,600	981,000
SMC-LSS	10,000	1	0.483	11	–
		2	0.111	12	–
		3	0.007	16	–
	100,000	1	0.547	139	–
		2	0.427	145	–
		3	0.590	159	–
	1,000,000	1	0.590	1,476	–
		2	0.587	1,446	–
		3	0.609	1,490	–

no strategy reaching the goal. We remark that deep RL has no issues with this track in the random start setting [46].

4.4 The *river-deadend-narrow* Track

Finally, we consider the *river-deadend-narrow* track (Fig. 4) in the shape of a river delta. The experimental results are shown in Table 5. The MDP of this model has 175,783,293 states. MCSTA can find a policy with a goal probability of 0.984 in 26 min (again slightly slowed down by swapping). As before, deep RL is not able to find a useful strategy in up to 30 min but delivers a nearly optimal strategy in around 10 h

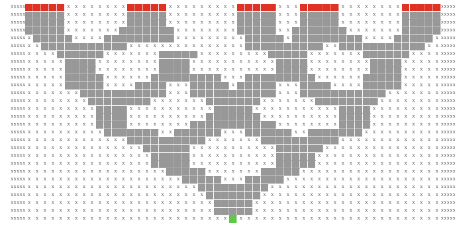


Fig. 4. The *river-deadend-narrow* track.

after 981,000 training episodes. SMC with LSS, for which we show the results for time bound 800 in Table 5 (since the time bound does not lead to a difference in results here), finds strategies that are successful up to 60 % of the time. While still far from the optimum, these are the best results that LSS achieves across the tracks we experiment with. We hypothesise that this is because no “complex navigation” is required to reach the goal; strategies that are mild variants of moving straight ahead are reasonably successful here. Of these, enough appear to exist for LSS to do reasonably well.

4.5 Summary

We observe that the Racetrack benchmarks are non-trivial in terms of decision-making, with the uniform random strategy being completely useless. MCSTA manages to analyse all of them with PMC, at the cost of white-box model access—and we intentionally selected tracks that are not too large to be able to make useful comparisons. LSS only works when the decisions need not be too specific. It appears to be an indicator of where learning, including deep RL with normal start, has difficulties starting up. Compared to deep RL, LSS is fast. Deep RL learned near-optimal strategies for all but the *maze* and the larger *barto* tracks, at a massive computational effort. In practice, the burden of this effort is on the GPU, making runtimes more acceptable. For the Racetrack cases, we could have significantly sped up deep RL (and made it work for all tracks) with the random start approach. Random start is clearly crucial for efficient deep RL; however it is only so easy to apply to intuitively structured models like Racetrack maps. For arbitrary verification models as, e.g., in the quantitative verification benchmark set [60], suitable random start procedures (e.g., by sensibly assigning non-initial values to the model’s variables) still need to be developed.

5 Outlook

Our survey of verification- and AI-based approaches for finding strategies highlights their very different characteristics in terms of the required model interface, memory usage, and runtime, which in turn depend on the structure of the MDP. We have seen that all three methods can be effective in finding reasonable, good, or even (near-)optimal strategies in suitable cases. All three methods also have tool support in the MODEST TOOLSET to apply them to verification models such as those from the quantitative verification benchmark set [60]. However, various challenges remain to make these approaches work better, interconnect them, and make the strategies they find useful and accessible to domain expert users.

Informed Exploration. We saw how crucial the random start process is to bootstrap the exploration phase in learning. For LSS, the uninformedness of the search is inherent: it simply picks many random strategies (in the sense of randomly chosen fixed decisions, *not* randomised decisions resampled every time as in the uniform random strategy). We speculate that, given a suitable heuristic indicating, e.g., the distance to the goal, the currently monolithic LSS strategies could be split into segments that could be individually sampled and combined, going backwards from the goal. A random start-like process for LSS may lead to *robust* strategies that work well not only for a single starting point.

Interconnecting Tools Through Strategies. Currently, specific connections exist between strategy-finding methods like LSS or deep RL and strategy-evaluation methods like SMC or PMC: the final phase of LSS is an SMC evaluation, and DSMC brings NN into MODES. Other works connect deep RL with PMC [25]. All of these are specific to a pair of strategy-finding and strategy-evaluation implementations. This is because today’s verification tools do not treat strategies as first-class objects. At the least, we should be able to apply a strategy found with any method to the model (determinising it) in the evaluation using any other method (such as PMC or SMC). This will require standardised formats or interfaces to represent or dynamically query strategies (similar to JANI), and further implementation work in tools to take strategies as input wherever possible.

Explaining Strategies. Each of the strategy-finding methods delivers its result in a specific format: lists of state-action pairs for PMC, an integer strategy identifier for LSS, and a NN definition for deep RL. None of them helps the user to understand and implement the strategy. Without understanding, especially in safety-critical situations, it is hard for users to trust the verification or AI tool’s result. A promising way out is to convert the strategy into an (often small and human-readable) decision tree, using as input an NN from deep RL [3] or a state-action pair list from PMC [6]. Ideally, we would integrate such a method directly into our verification and learning tool ecosystem around the MODEST TOOLSET, which poses technical but also conceptual challenges to, e.g., obtain a decision tree from an LSS scheduler identifier without having to exhaustively

enumerate all of the MDP’s states and thus negating the memory advantage of the SMC-LSS approach. And in the end, while such approaches may make the strategy understandable, they still do not explain why the strategy at hand is (near-)optimal and should be the one to be implemented. A different approach is visualisation, which works well for illustrative benchmarks like Racetrack or other cyber-physical systems. A first such tool is TraceVis [43], visualising the DSMC results and traces. An extension of the tool towards visualisation of the NN internals and the learning process is currently under development.

Beyond Discrete Markov Models. Models with general probability distributions, such as stochastic automata [31], are desirable to more realistically represent phenomena such as time-to-failure distributions or combinations of failures, inspections, repairs, and attacks. Continuous dynamics specified by differential equations, as in hybrid automata [65], allow the inclusion of models of physical processes and thus the analysis of cyber-physical systems. SMC also effectively works for non-Markovian and hybrid formalisms, as evidenced by MODES’ support for stochastic hybrid automata, however LSS does not [30]. We have recently combined SMC for such models with RL, but used explicitly stored Q-functions and discretisation for learning [86]. PMC approaches for such models are the subject of active research, with simple approaches based on interval abstractions provided by the MODEST TOOLSET today [53, 54]. NN, on the other hand, can in principle handle continuous inputs and outputs just as well as discrete ones. In light of these various advances of learning and verification into non-Markovian continuous-time and continuous-state models, a coherent toolchain that supports verification models with a focus on strategies is still lacking today.

Future Racetracks. We are working on a continuous version of the Racetrack benchmark where the car does not move in a discrete grid. We also continuously extend the benchmark with features like tanks to restrict fuel consumption, different engine types, and other variants [9]. The current developments of Race-track can always be found online at racetrack.perspicuous-computing.science.

Data Availability. A dataset with models, tools, and scripts from our experimental evaluation is archived and available at DOI <https://doi.org/10.4121/20669646>.

Acknowledgments. The authors thank Timo P. Gros and Maximilian A. Köhl for their work on the tools we build on.

References

1. Agha, G., Palmeskog, K.: A survey of statistical model checking. *ACM Trans. Model. Comput. Simul.* **28**(1), 6:1–6:39 (2018). <https://doi.org/10.1145/3158668>
2. Agostinelli, F., McAleer, S., Shmakov, A., Baldi, P.: Solving the Rubik’s cube with deep reinforcement learning and search. *Nat. Mach. Intell.* **1**, 356–363 (2019)

3. Alamdari, P.A., Avni, G., Henzinger, T.A., Lukina, A.: Formal methods with a touch of magic. In: 2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, 21–24 September 2020, pp. 138–147. IEEE (2020). https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_21
4. Ashok, P., Butkova, Y., Hermanns, H., Křetínský, J.: Continuous-time Markov decisions based on partial exploration. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 317–334. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_19
5. Ashok, P., Daca, P., Křetínský, J., Weininger, M.: Statistical model checking: black or white? In: Margaria, T., Steffen, B. (eds.) ISO/LA 2020. LNCS, vol. 12476, pp. 331–349. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61362-4_19
6. Ashok, P., Jackermeier, M., Křetínský, J., Weinhuber, C., Weininger, M., Yadav, M.: dtControl 2.0: explainable strategy representation via decision tree learning steered by experts. In: Groote, J.F., Larsen, K.G. (eds.) TACAS 2021. LNCS, vol. 12652, pp. 326–345. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72013-1_17
7. Avni, G., Bloem, R., Chatterjee, K., Henzinger, T.A., Könighofer, B., Pranger, S.: Run-time optimization for learned controllers through quantitative games. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 630–649. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_36
8. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 963–999. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_28
9. Baier, C., et al.: Lab conditions for research on explainable automated decisions. In: Heintz, F., Milano, M., O’Sullivan, B. (eds.) TAILOR 2020. LNCS (LNAI), vol. 12641, pp. 83–90. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-73959-1_8
10. Bard, N., et al.: The Hanabi challenge: a new frontier for AI research. *Artif. Intell.* **280**, 103216 (2020)
11. Barto, A.G., Bradtko, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. *Artif. Intell.* **72**(1–2), 81–138 (1995)
12. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: an evaluation platform for general agents. *J. Artif. Intell. Res.* **47**, 253–279 (2013)
13. Bellman, R.: A Markovian decision process. *J. Math. Mech.* **6**(5), 679–684 (1957)
14. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60692-0_70
15. Bogdoll, J., Ferrer Fioriti, L.M., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: Bruni, R., Dingel, J. (eds.) FMOODS/FORTE -2011. LNCS, vol. 6722, pp. 59–74. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21461-5_4
16. Bohnenkamp, H.C., D’Argenio, P.R., Hermanns, H., Katoen, J.: MODEST: a compositional modeling formalism for hard and softly timed systems. *IEEE Trans. Software Eng.* **32**(10), 812–830 (2006). <https://doi.org/10.1109/TSE.2006.104>
17. Bonet, B., Geffner, H.: Labeled RTDP: improving the convergence of real-time dynamic programming. In: ICAPS, pp. 12–21 (2003)

18. Brázdil, T., et al.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_8
19. Brockman, G., et al.: OpenAI gym. CoRR [arXiv:abs/1606.01540](https://arxiv.org/abs/1606.01540) (2016)
20. Budde, C.E., D’Argenio, P.R., Hartmanns, A.: Automated compositional importance splitting. *Sci. Comput. Program.* **174**, 90–108 (2019). <https://doi.org/10.1016/j.scico.2019.01.006>
21. Budde, C.E., D’Argenio, P.R., Hartmanns, A., Sedwards, S.: An efficient statistical model checker for nondeterminism and rare events. *Int. J. Softw. Tools Technol. Transf.* **22**(6), 759–780 (2020). <https://doi.org/10.1007/s10009-020-00563-2>
22. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 151–168. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_9
23. Budde, C.E., et al.: On correctness, precision, and performance in quantitative verification. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020. LNCS, vol. 12479, pp. 216–241. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-83723-5_15
24. Butkova, Y., Hartmanns, A., Hermanns, H.: A Modest approach to Markov automata. *ACM Trans. Model. Comput. Simul.* **31**(3), 14:1–14:34 (2021). <https://doi.org/10.1145/3449355>
25. Carr, S., Jansen, N., Wimmer, R., Serban, A.C., Becker, B., Topcu, U.: Counterexample-guided strategy improvement for POMDPs using recurrent neural networks. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, 10–16 August 2019, pp. 5532–5539 (2019). <https://doi.org/10.24963/ijcai.2019/768>. <https://www.ijcai.org/>
26. Chung, T.H., Burdick, J.W.: A decision-making framework for control strategies in probabilistic search. In: 2007 IEEE International Conference on Robotics and Automation, ICRA 2007, Roma, Italy, 10–14 April 2007, pp. 4386–4393. IEEE (2007). <https://doi.org/10.1109/ROBOT.2007.364155>
27. Côté, M.-A., et al.: TextWorld: a learning environment for text-based games. In: Cazenave, T., Saffidine, A., Sturtevant, N. (eds.) CGW 2018. CCIS, vol. 1017, pp. 41–75. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24337-1_3
28. Dai, P., Mausam, Weld, D.S., Goldsmith, J.: Topological value iteration algorithms. *J. Artif. Intell. Res.* **42**, 181–209 (2011). <http://jair.org/papers/paper3390.html>
29. D’Argenio, P.R., Fraire, J.A., Hartmanns, A.: Sampling distributed schedulers for resilient space communication. In: Lee, R., Jha, S., Mavridou, A., Giannakopoulou, D. (eds.) NFM 2020. LNCS, vol. 12229, pp. 291–310. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-55754-6_17
30. D’Argenio, P.R., Hartmanns, A., Sedwards, S.: Lightweight statistical model checking in nondeterministic continuous time. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 336–353. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03421-4_22
31. D’Argenio, P.R., Katoen, J.P.: A theory of stochastic systems part I: stochastic automata. *Inf. Comput.* **203**(1), 1–38 (2005). <https://doi.org/10.1016/j.ic.2005.07.001>
32. D’Argenio, P.R., Legay, A., Sedwards, S., Traonouez, L.M.: Smart sampling for lightweight verification of Markov decision processes. *Int. J. Softw. Tools Technol. Transf.* **17**(4), 469–484 (2015). <https://doi.org/10.1007/s10009-015-0383-0>

33. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: UPPAAL STRATEGO. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 206–211. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_16
34. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_27
35. Doshi-Velez, F., Kim, B.: Towards a rigorous science of interpretable machine learning. arXiv preprint [arXiv:1702.08608](https://arxiv.org/abs/1702.08608) (2017)
36. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, Edinburgh, UK, 11–14 July 2010, pp. 342–351. IEEE Computer Society (2010). <https://doi.org/10.1109/LICS.2010.41>
37. Fan, L., et al.: Surreal: open-source reinforcement learning framework and robot manipulation benchmark. In: Conference on Robot Learning, pp. 767–782. PMLR (2018)
38. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 53–113. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21455-4_3
39. François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J.: An introduction to deep reinforcement learning. *Found. Trends Mach. Learn.* **11**(3–4), 219–354 (2018). <https://doi.org/10.1561/22000000071>
40. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and safety verification for stochastic hybrid systems. In: Caccamo, M., Frazzoli, E., Grosu, R. (eds.) Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, IL, USA, 12–14 April 2011, pp. 43–52. ACM (2011). <https://doi.org/10.1145/1967701.1967710>
41. Gardner, M.: Mathematical games. *Sci. Am.* **229**, 118–121 (1973)
42. Gribaudo, M., Remke, A.: Hybrid Petri nets with general one-shot transitions. *Perform. Eval.* **105**, 22–50 (2016). <https://doi.org/10.1016/j.peva.2016.09.002>
43. Gros, T.P., Groß, D., Gumhold, S., Hoffmann, J., Klauck, M., Steinmetz, M.: TraceVis: towards visualization for deep statistical model checking. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020. LNCS, vol. 12479, pp. 27–46. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-83723-5_3
44. Gros, T.P., Hermanns, H., Hoffmann, J., Klauck, M., Steinmetz, M.: Deep statistical model checking. In: Gotsman, A., Sokolova, A. (eds.) FORTE 2020. LNCS, vol. 12136, pp. 96–114. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-50086-3_6
45. Gros, T.P., Hermanns, H., Hoffmann, J., Klauck, M., Köhl, M.A., Wolf, V.: MoGym: using formal models for training and verifying decision-making agents. In: CAV 2022 (2022, to appear)
46. Gros, T.P., Höller, D., Hoffmann, J., Klauck, M., Meerkamp, H., Wolf, V.: DSMC evaluation stages: fostering robust and safe behavior in deep reinforcement learning. In: Abate, A., Marin, A. (eds.) QEST 2021. LNCS, vol. 12846, pp. 197–216. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85172-9_11
47. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning, pp. 1861–1870. PMLR (2018)

48. Haddad, S., Monmege, B.: Reachability in MDPs: refining convergence of value iteration. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) RP 2014. LNCS, vol. 8762, pp. 125–137. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11439-2_10
49. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. *Theor. Comput. Sci.* **735**, 111–131 (2018). <https://doi.org/10.1016/j.tcs.2016.12.003>
50. Hahn, E.M., Hartmanns, A.: A comparison of time- and reward-bounded probabilistic model checking techniques. In: Fränzle, M., Kapur, D., Zhan, N. (eds.) SETTA 2016. LNCS, vol. 9984, pp. 85–100. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47677-3_6
51. Hahn, E.M., Hartmanns, A.: Symblicit exploration and elimination for probabilistic model checking. In: Hung, C., Hong, J., Bechini, A., Song, E. (eds.) The 36th ACM/SIGAPP Symposium on Applied Computing, SAC 2021, Virtual Event, Republic of Korea, 22–26 March 2021, pp. 1798–1806. ACM (2021). <https://doi.org/10.1145/3412841.3442052>
52. Hahn, E.M., et al.: The 2019 comparison of tools for the analysis of quantitative formal models. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) TACAS 2019. LNCS, vol. 11429, pp. 69–92. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17502-3_5
53. Hahn, E.M., Hartmanns, A., Hermanns, H.: Reachability and reward checking for stochastic timed automata. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* **70**, 1–15 (2014). <https://doi.org/10.14279/tuj.eceasst.70.968>
54. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.: A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods Syst. Des.* **43**(2), 191–232 (2013). <https://doi.org/10.1007/s10703-012-0167-z>
55. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Faithful and effective reward schemes for model-free reinforcement learning of omega-regular objectives. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 108–124. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_6
56. Hartmanns, A.: Correct probabilistic model checking with floating-point arithmetic. In: Fisman, D., Rosu, G. (eds.) TACAS 2022. LNCS, vol. 13244, pp. 41–59. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99527-0_3
57. Hartmanns, A., Hermanns, H.: The Modest Toolset: an integrated environment for quantitative modelling and verification. In: Abraham, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 593–598. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_51
58. Hartmanns, A., Hermanns, H.: Explicit model checking of very large MDP using partitioning and secondary storage. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 131–147. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24953-7_10
59. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 488–511. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53291-8_26
60. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 344–350. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_20

61. Hartmanns, A., Timmer, M.: Sound statistical model checking for MDP using partial order and confluence reduction. *Int. J. Softw. Tools Technol. Transf.* **17**(4), 429–456 (2015). <https://doi.org/10.1007/s10009-014-0349-7>
62. Hasanbeig, M., Abate, A., Kroening, D.: Logically-correct reinforcement learning. *CoRR arXiv:1801.08099* (2018)
63. Henriques, D., Martins, J.G., Zuliani, P., Platzer, A., Clarke, E.M.: Statistical model checking for Markov decision processes. In: Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, UK, 17–20 September 2012, pp. 84–93. IEEE Computer Society (2012). <https://doi.org/10.1109/QEST.2012.19>
64. Hensel, C., Junges, S., Katoen, J.P., Quatmann, T., Volk, M.: The probabilistic model checker Storm. *Int. J. Softw. Tools Technol. Transf.* (2021). <https://doi.org/10.1007/s10009-021-00633-z>
65. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings of 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, 27–30 July 1996, pp. 278–292. IEEE Computer Society (1996). <https://doi.org/10.1109/LICS.1996.561342>
66. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24622-0_8
67. Hinton, G., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal Process. Mag.* **29**(6), 82–97 (2012)
68. Ho, J., Ermon, S.: Generative adversarial imitation learning. *Adv. Neural. Inf. Process. Syst.* **29**, 4565–4573 (2016)
69. Howard, R.A.: *Dynamic Probabilistic Systems: Semi-Markov and Decision Processes*. Dover Books on Mathematics, vol. 2. Dover Publications, Mineola (2013)
70. Howard, R.A.: *Dynamic Programming and Markov Processes*. MIT Press, Cambridge (1960)
71. Junges, S., Jansen, N., Dehnert, C., Topcu, U., Katoen, J.-P.: Safety-constrained reinforcement learning for MDPs. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 130–146. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_8
72. Keller, T., Eyerich, P.: PROST: probabilistic planning based on UCT. In: McCluskey, L., Williams, B.C., Silva, J.R., Bonet, B. (eds.) Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, 25–29 June 2012. AAAI (2012). <http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4715>
73. Klauck, M., Hermanns, H.: A Modest approach to dynamic heuristic search in probabilistic model checking. In: Abate, A., Marin, A. (eds.) QEST 2021. LNCS, vol. 12846, pp. 15–38. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85172-9_2
74. Köhl, M.A., Klauck, M., Hermanns, H.: Momba: JANI meets Python. In: TACAS 2021. LNCS, vol. 12652, pp. 389–398. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72013-1_23
75. Kolobov, A., Mausam, Weld, D.S., Geffner, H.: Heuristic search for generalized stochastic shortest path MDPs. In: Bacchus, F., Domshlak, C., Edelkamp, S., Helmert, M. (eds.) Proceedings of the 21st International Conference on

- Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany, 11–16 June 2011. AAAI (2011). <http://aaai.org/ocs/index.php/ICAPS/ICAPS11/paper/view/2682>
76. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: NIPS, pp. 1097–1105 (2012)
 77. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72522-0_6
 78. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
 79. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. *Theor. Comput. Sci.* **282**(1), 101–150 (2002). [https://doi.org/10.1016/S0304-3975\(01\)00046-9](https://doi.org/10.1016/S0304-3975(01)00046-9)
 80. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: an overview. In: Barringer, H., et al. (eds.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16612-9_11
 81. Legay, A., Sedwards, S., Traonouez, L.-M.: Scalable verification of Markov decision processes. In: Canal, C., Idani, A. (eds.) SEFM 2014. LNCS, vol. 8938, pp. 350–362. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15201-1_23
 82. McMahan, H.B., Gordon, G.J.: Fast exact planning in Markov decision processes. In: ICAPS, pp. 151–160 (2005)
 83. McMahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: Raedt, L.D., Wrobel, S. (eds.) Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, 7–11 August 2005. ACM International Conference Proceeding Series, vol. 119, pp. 569–576. ACM (2005). <https://doi.org/10.1145/1102351.1102423>
 84. Mnih, V., et al.: Playing Atari with deep reinforcement learning. arXiv preprint [arXiv:1312.5602](https://arxiv.org/abs/1312.5602) (2013). Accessed 15 Sept 2020
 85. Mnih, V., et al.: Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015)
 86. Niehage, M., Hartmanns, A., Remke, A.: Learning optimal decisions for stochastic hybrid systems. In: Arun-Kumar, S., Méry, D., Saha, I., Zhang, L. (eds.) 19th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2021, Virtual Event, China, 20–22 November 2021, pp. 44–55. ACM (2021). <https://doi.org/10.1145/3487212.3487339>
 87. Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. *Ann. Inst. Stat. Math.* **10**(1), 29–35 (1959)
 88. Parker, D.A.: Implementation of symbolic model checking for probabilistic systems. Ph.D. thesis, University of Birmingham, UK (2003)
 89. Pathak, D., Agrawal, P., Efros, A.A., Darrell, T.: Curiosity-driven exploration by self-supervised prediction. In: International Conference on Machine Learning, pp. 2778–2787. PMLR (2017)
 90. Pineda, L.E., Lu, Y., Zilberstein, S., Goldman, C.V.: Fault-tolerant planning under uncertainty. In: IJCAI, pp. 2350–2356 (2013)
 91. Pineda, L.E., Zilberstein, S.: Planning under uncertainty using reduced models: revisiting determinization. In: ICAPS (2014)
 92. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York (1994)

93. Quatmann, T., Katoen, J.-P.: Sound value iteration. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 643–661. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_37
94. Reijsbergen, D., de Boer, P., Scheinhardt, W.R.W., Haverkort, B.R.: On hypothesis testing for statistical model checking. *Int. J. Softw. Tools Technol. Transf.* **17**(4), 377–395 (2015). <https://doi.org/10.1007/s10009-014-0350-1>
95. Rubino, G., Tuffin, B. (eds.): *Rare Event Simulation Using Monte Carlo Methods*. Wiley, New York (2009). <https://doi.org/10.1002/9780470745403>
96. Sarle, W.S.: *Neural networks and statistical models* (1994)
97. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017)
98. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* **362**(6419), 1140–1144 (2018)
99. Sproston, J.: Decidable model checking of probabilistic hybrid automata. In: Joseph, M. (ed.) FTRTFT 2000. LNCS, vol. 1926, pp. 31–45. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45352-0_5
100. Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning, 2nd edn. The MIT Press, Cambridge (2018)
101. Wald, A.: Sequential tests of statistical hypotheses. *Ann. Math. Stat.* **16**(2), 117–186 (1945)
102. Waschneck, B., et al.: Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP* **72**, 1264–1269 (2018)
103. Xia, F., Zamir, A.R., He, Z., Sax, A., Malik, J., Savarese, S.: Gibson env: real-world perception for embodied agents. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9068–9079 (2018)
104. Yoon, S.W., Fern, A., Givan, R.: FF-replan: a baseline for probabilistic planning. In: Boddy, M.S., Fox, M., Thiébaux, S. (eds.) *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, 22–26 September 2007*, p. 352. AAAI (2007). <http://www.aaai.org/Library/ICAPS/2007/icaps07-045.php>
105. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_17
106. Yu, T., et al.: Meta-world: a benchmark and evaluation for multi-task and meta reinforcement learning. In: *Conference on Robot Learning*, pp. 1094–1100. PMLR (2020)