Robust Anytime Learning of Markov Decision Processes

Anonymous Author(s)
Affiliation
Address
email

Abstract

Markov decision processes (MDPs) are formal models commonly used in sequential decision-making. MDPs capture the stochasticity that may arise, for instance, from imprecise actuators via probabilities in the transition function. However, in data-driven applications, deriving precise probabilities from (limited) data introduces statistical errors that may lead to unexpected or undesirable outcomes. Uncertain MDPs (uMDPs) do not require precise probabilities but instead use so-called uncertainty sets in the transitions, accounting for such limited data. Tools from the formal verification community efficiently compute robust policies that provably adhere to formal specifications, like safety constraints, under the worst-case instance in the uncertainty set. We continuously learn the transition probabilities of an MDP in a robust anytime-learning approach that combines a dedicated Bayesian inference scheme with the computation of robust policies. In particular, our method (1) approximates probabilities as intervals, (2) adapts to new data that may be inconsistent with an intermediate model, and (3) may be stopped at any time to compute a robust policy on the uMDP that faithfully captures the data so far. We show the effectiveness of our approach and compare it to robust policies computed on uMDPs learned by the UCRL2 reinforcement learning algorithm in an experimental evaluation on several benchmarks.

1 Introduction

Sequential decision-making in realistic scenarios is inherently subject to uncertainty, commonly captured via probabilities. *Markov decision processes* (MDPs) are the standard model to reason about such decision-making problems [Puterman, 1994, Bertsekas, 2005]. Safety-critical scenarios require assessments of correctness which can, for instance, be described by temporal logic [Pnueli, 1977] or expected reward specifications. A fundamental requirement for providing such correctness guarantees on MDPs is that probabilities are precisely given. Methods such as variants of model-based reinforcement learning [Moerland et al., 2020] or PAC-learning [Strehl et al., 2009] can learn MDPs by deriving *point estimates* of probabilities from data to satisfy this requirement. This derivation naturally carries the risk of statistical errors. Optimal policies are highly sensitive to small perturbations in transition probabilities, leading to sub-optimal outcomes such as a deterioration in performance [Mannor et al., 2007, Goyal and Grand-Clement, 2020].

Uncertain MDPs (uMDPs) extend MDPs to incorporate such statistical errors by introducing an additional layer of uncertainty via *uncertainty sets* on the transition function [Nilim and Ghaoui, 2005, Wiesemann et al., 2013, Goyal and Grand-Clement, 2020]. The solution of a uMDP is a *robust policy* that allows an adversarial selection (i.e. the worst-case scenario) of probabilities within the uncertainty set, and induces a *worst-case performance* (a conservative bound on, e.g., the reachability probability or expected reward). The problem of computing such robust policies, also called *robust*

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

verification, is solved using value iteration or convex optimization, where the uncertainty sets are convex [Wolff et al., 2012, Puggelli et al., 2013].

Our approach. We study the problem of learning an MDP from data. We propose an iterative learning method which uses uMDPs as intermediate models and is able to *adapt to new data* which may be inconsistent with prior assumptions. Furthermore, the method is *task-aware* in the sense that the learning procedure respects temporal logic specifications. In particular, our method learns intervals of probabilities for individual transitions. This Bayesian *anytime-learning approach* employs intervals with linearly updating conjugate priors [Walter and Augustin, 2009], and can iteratively improve upon a uMDP that approximates the true MDP we wish to learn. This method not only decreases the size of each interval, but may also increase it again in case of a so-called *prior-data conflict* where new data suggests the actual probability lies outside the current interval. Consequently, a newly learned interval does not need to be a subset of its prior interval. Alternatively, we also include *probably approximately correct* (PAC) intervals via Hoeffding's inequality [Hoeffding, 1963], which introduces a correctness guarantee for each transition.

We summarize the key features of our learning method, and what sets it apart from other methods.

- *An anytime approach.* The ability to iteratively update intervals that are not necessarily subsets of each other allows us to design an *anytime-learning approach*. At any time, we may stop the learning and compute a robust policy for the uMDP that the process has yielded thus far, together with the worst-case performance of this policy against a given specification. This performance may not be satisfactory, e.g., the worst-case probability to reach a set of critical states may be below a certain threshold. In this case, we continue learning towards a new uMDP that more faithfully captures the true MDP due to the inclusion of further data. Thereby, we ensure that the robust policy gradually gets closer to the optimal policy for the true MDP.
- *Specification-driven.* Our method features the possibility to learn in a task-aware fashion, that is, to learn transitions that matter for a given specification. In particular, for reachability or expected reward (temporal logic) specifications which require a certain set of target states to be reached, we only learn and update transitions along paths towards these states. Transitions outside those paths do not affect the satisfaction of the specification.
- *Flexibility in interval computation.* Our method can easily be adapted to different methods for computing the intervals. Besides the linearly updating intervals, we also implement PAC intervals, and compare to the UCRL2 algorithm [Jaksch et al., 2010].

2 Related Work

Reinforcement learning (RL). Uncertain MDPs have (often implicitly) been used by RL algorithms, for instance, to mitigate the *exploration/exploitation* trade-off by guiding the RL agent towards unexplored parts of the environment, following the *optimism in the face of uncertainty* principle [Jaksch et al., 2010, Fruit et al., 2017]. We use the same principle in the exploration phase of our procedure, but compute robust policies as output to account for the uncertainty in an adversarial (or pessimistic) way. Similarly, uncertain MDPs are used to compute robust policies when the data available is limited [Nilim and Ghaoui, 2005, Wiesemann et al., 2013, Russel and Petrik, 2019]. Such robustness is connected to a *pessimistic* principle that has been effective in offline RL settings [Lange et al., 2012], where the agent only has access to a fixed dataset of past trajectories, meaning it needs to base decisions on limited data [Rashidinejad et al., 2021, Buckman et al., 2021, Jin et al., 2021]. Likewise, our method may stop and return a policy before the problem is fully explored.

Robust RL concerns the standard RL problem, but explicitly accounts for input disturbances and model errors [Morimoto and Doya, 2005]. Often there is a focus on ensuring that a reasonable performance is achieved during data collection. To that end, Lim et al. [2013] and Derman et al. [2019] sample trajectories using a robust policy, which can slow down the process to find an optimal policy. We assume sampling access to the underlying MDP, which allows us to use more efficient exploration. It should be noted that we only use uMDPs as an intermediate model towards learning a standard MDP, whereas in robust RL the model itself may also be an (adversarial) uncertain MDP. As a result, our method converges to an MDP, while robust RL attempts to learn and possibly converge to an uncertain MDP.

The problem of learning an MDP from data is also related to *model learning*, which typically assumes no knowledge about the states and thus iteratively increases a the set of states [Vaandrager, 2017]. Work on model learning probabilistic systems such as MDPs is as of yet sparse. In [Tappler et al., 2019] the classic L^* algorithm for learning finite automata is adapted for MDPs. This algorithm only yields point estimates of probabilities and makes strong assumptions on the structure of the MDP that is being learned.

Ashok et al. [2019] use PAC-learning to estimate the transition functions of MDPs and stochastic games in order to perform *statistical model checking*. There, the PAC-bounds are used to construct intervals and then the resulting model is used for statistical model checking. In contrast to our method, they only learn the model once and do not iteratively update in the presence of new data.

Finally, literature distinguishes two types of uncertainty: *aleatoric* and *epistemic* uncertainty [Hüllermeier and Waegeman, 2021]. Aleatoric uncertainty refers to the uncertainty generated by a probability distribution, like the transition function of an MDP, and is also known as *irreducible uncertainty*. In contrast, epistemic uncertainty is *reducible* by collecting and accounting for (new) data. Our work can be seen as adding an additional layer of epistemic uncertainty on the probability distributions of the transition function that is then, by gathering and including more data, reduced.

3 Preliminaries

A *discrete probability distribution* over a finite set X is a function $\mu\colon X \to [0,1] \subset \mathbb{R}$ with $\sum_{x \in X} \mu(x) = 1$. We write $\mathcal{D}(X)$ for the set of all discrete probability distributions over X, and by $|X|$ we denote the number of elements in X. For any interval $I \subseteq \mathbb{R}$ we write $\underline{I}$ and $\overline{I}$ for the lower and upper bounds of the interval, that is, $I = [\underline{I}, \overline{I}]$.

Definition 1 (Markov decision process)**.** *A* Markov decision process *(MDP) is a tuple* (S, s_I, A, P, R) *with* S *a finite set of states,* $s_I \in S$ *the initial state,* A *a finite set of actions,* $P\colon S \times A \times S \to [0,1]$ *with* $\forall s, a \in S \times A, \sum_{s'} P(s,a,s') = 1$ *(such that* $P(s,a) \in \mathcal{D}(S)$*) the probabilistic transition function, and* $R\colon S \times A \to \mathbb{R}_{\geq 0}$ *the reward (or cost) function.*

A *trajectory* in an MDP is a finite sequence $(s_0, a_0, s_1, a_1, \ldots, s_n) \in (S \times A)^* \times S$ where $s_0 = s_I$ and $P(s_i, a_i, s_{i+1}) > 0$ for $0 \leq i < n$. A *deterministic memoryless* (or pure) policy is a function $\pi\colon S \to A$. Applying a policy to an MDP M resolves all the non-deterministic choices and yields an (induced) *discrete-time Markov chain* (DTMC), see Baier and Katoen [2008] for details.

Definition 2 (Uncertain MDP)**.** *An* uncertain MDP (uMDP) *is a tuple* $(S, s_I, A, \mathbb{I}, \mathcal{P}, R)$ *where* S, s_I, A, R *are as for MDPs,* $\mathbb{I}$ *is a set of probability intervals* $\mathbb{I} = \{[a,b] \mid 0 < a \leq b \leq 1\}$*, and* $\mathcal{P}\colon S \times A \times S \to (\mathbb{I} \cup \{0\})$ *is the* uncertain transition function*, assigning either a probability interval, or the exact probability* 0 *to any transition.*

Uncertain MDPs can be seen as an uncountable set of MDPs that only differ in their transition functions. For a transition function P, we write $P \in \mathcal{P}$ if for every transition the probability of P lies within the interval of $\mathcal{P}$, i.e., $P(s,a,s') \in \mathcal{P}(s,a,s')$ for all $(s,a,s') \in S \times A \times S$. We only allow intervals with a lower bound greater than zero, to ensure a transition cannot vanish under certain distributions generated by the uncertainty. This assumption is standard and required for robust verification [Wiesemann et al., 2013, Puggelli et al., 2013].

Specifications. We consider *reachability* or *expected reward (cost)* specifications. The value $\mathbb{P}_\pi^M(\Diamond T)$ is the probability to reach a set of *target* states $T \subseteq S$ on the MDP M under the policy π, also referred to as the *performance of* π[1]. Likewise, $\mathrm{R}_\pi^M(\Diamond T)$ describes the expected accumulated reward to reach T under π. Note that for probabilistic specifications, we can replace the formula $\Diamond T$ by more general temporal logic formulas, since they reduce to reachability via the standard product construction [Baier and Katoen, 2008].

Formally, the specification $\mathbb{P}_{\text{Max}}(\Diamond T) = \max_\pi \mathbb{P}_\pi^M(\Diamond T)$ expresses that the probability of eventually reaching the target set $T \subseteq S$ should be maximal. Likewise, the specification $\mathrm{R}_{\text{Max}}(\Diamond T)$ requires the expected reward for reaching T to be maximal. For minimization, we write $\mathbb{P}_{\text{Min}}(\Diamond T)$ and $\mathrm{R}_{\text{Min}}(\Diamond T)$, respectively. Besides optimizing a probability or reward, a specification may also express an explicit user-provided threshold to compare the performance of a policy to.

[1] We will omit the policy π and the MDP M whenever they are clear from the context.

For uMDPs, we define *optimistic* and *pessimistic* specifications. In optimistic specifications, we assume the best-case scenario of the uncertainty to satisfy the specification by also minimizing (or maximizing) over the uncertainty set, written as $\mathbb{P}_{\text{MinMin}}(\Diamond T) = \min_\pi \min_{P \in \mathcal{P}} \mathbb{P}_\pi^{\mathcal{M}[P]}(\Diamond T)$ (or $\mathbb{P}_{\text{MaxMax}}(\Diamond T)$), where the first Min (Max) signals what the decision-maker is trying to achieve, and the second what the uncertainty does. In pessimistic specifications, the uncertainty does the opposite of the goal: $\mathbb{P}_{\text{MaxMin}}(\Diamond T)$ or $\mathbb{P}_{\text{MinMax}}(\Diamond T)$. The notation is similar for reward specifications. A (standard) specification φ can be extended to be optimistic or pessimistic by adding a second Min or Max. We write φ_O for its optimistic extension, and φ_P for its pessimistic extension.

For an MDP M, the aim is to compute a policy π that either optimizes a given specification φ, or whose performance respects a given threshold that, e.g., provides an upper bound on the probability of reaching a set of critical states. Common methods are value iteration or linear programming [Puterman, 1994, Baier and Katoen, 2008]. For uncertain MDPs $\mathcal{M}$, the goal is to compute a policy π that satisfies an optimistic or pessimistic specification φ_O or φ_P. In the latter case, we call π a *robust policy*. Optimal policies in uMDPs can be computed via robust dynamic programming or convex optimization [Wolff et al., 2012, Puggelli et al., 2013].

4 Problem Statement and Procedure Outline

We have an unknown but fixed MDP $M = (S, s_I, A, P, R)$, which we will refer to as the *true MDP*, an *initial prior uMDP* $\mathcal{M} = (S, s_I, A, \mathbb{I}, \mathcal{P}, R)$, and a specification φ which we want to satisfy. A discussion of prior (and other parameter) choices follows in Section 6.

Assumption 1 (Underlying graph)**.** *We assume that the underlying graph of the true MDP is known. In particular: transitions that do not exist in the true MDP (transitions of probability* 0*) do also not exist in the uMDP, transitions of probability* 1 *in the true MDP are assigned the point interval* $[1, 1]$ *in the uMDP, and any other transition of non-zero probability* p *has an interval* $I \in \mathbb{I}$ *in the uMDP.*

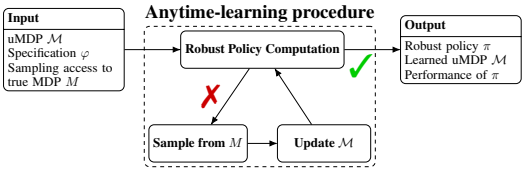


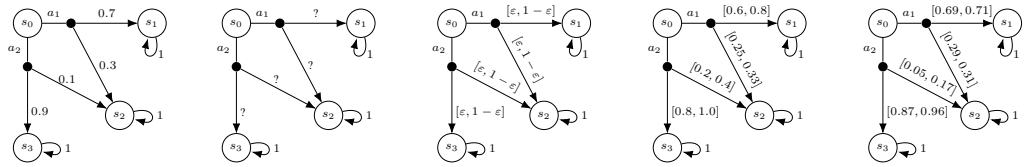
Figure 1: Procedure outline.

Under Assumption 1 and Definition 2, we construct the initial prior uMDP $\mathcal{M}$ to have transitions of probability 0 and 1 exactly where the true MDP M has these too, and interval transitions $[\varepsilon, 1 - \varepsilon]$ for all other transitions, with $\varepsilon > 0$ free to choose. In particular, our approach does not require ε to be smaller than the smallest probability $p > 0$ occurring in M, which we also do not assume to be known. Alternatively, in case further knowledge is available, one may use any other prior uMDP as long as it satisfies Assumption 1. Our learning problem is expressed as follows:

> The problem is to learn the transition probabilities of a true MDP M, driven by a specification φ, via intermediate uncertain MDPs $\mathcal{M}$ that are iteratively updated to account for newly collected data, such that at any time a robust policy can be computed.

In the following we outline our anytime-learning procedure as illustrated in Figure 1.

1. **Input.** We start with an initial prior uMDP $\mathcal{M}$ and a (temporal logic) specification φ we wish to verify. Furthermore, we assume access to the unknown *true MDP* M to sample from. Alternatively, we may also assume a (constant) stream of observations from the true MDP.
2. **Robust policy computation.** We compute a robust policy π for the pessimistic extension of φ, i.e. φ_P, in the uncertain MDP $\mathcal{M}$, together with the worst-case performance of the specification in the current uMDP. If the specification contains an explicit threshold, the value can be compared against the threshold for automatic termination. In case of specifications that optimize a probability or reward, termination needs to be done manually, as it is impossible to tell if the maximum (minimum) was achieved.
3. **Anytime learning.** If the result from step 2 is unsatisfactory, we starts learning:
   (a) **Exploration.** We sample one or more trajectories from the true MDP M, using the *optimism in the face of uncertainty* principle.



(a) The true MDP. (b) Graph assumption. (c) Initial prior uMDP for $\varepsilon > 0$. (d) Intermediate learned uMDP. (e) Learned uMDP converging towards the MDP.

Figure 2: Process flow on an example MDP.

(b) **Update.** We update the intervals of the uMDP $\mathcal{M}$ in accordance with the newly collected data. This update yields a new uMDP that more faithfully captures all collected data up to this point than the previous uMDP.

(c) **Repeat.** We start again at step 2 with this new uMDP until (manual) termination.

4. **Output.** The process may be stopped at any moment and yields the latest uMDP $\mathcal{M}$ together with robust policy π and the performance of π on $\mathcal{M}$.

The effects of how this procedure could work out are illustrated in Figure 2. In 2a, we see an example MDP M to learn, and 2b shows the assumed knowledge about M. 2c shows the initial uMDP $\mathcal{M}$ constructed from 2b, using a user-provided lower bound $\varepsilon > 0$ to ensure that all lower bounds of $\mathcal{M}$ are strictly greater than zero. In 2d, we see an intermediate learned uMDP. Some intervals may already have successfully converged towards the probability of that transition in the true MDP, while others may be very inaccurate due to a low sample size and thus a bad estimate. Finally, 2e depicts the learned uMDP converging towards the true MDP.

5 Bayesian Learning

We assume access to trajectories through the true MDP. Due to the Markov property of the MDP, i.e., the fact that the transition probabilities only depend on the current state and not on any further history, we may split each trajectory τ into separate sets of independent experiments where a state-action pair (s, a) is sampled and a successor state s_i is observed, see also Appendix A of [Strehl and Littman, 2008]. We count the number of occurrences of the transition (s, a, s_i) in each trajectory τ, and the number of occurrences of the state-action pair (s, a) in each trajectory τ, denoted by $\#(s, a, s_i)$ and $\#(s, a)$, respectively. In the following, we introduce the two approaches of learning intervals: via Hoeffding's inequality, which provides a PAC guarantee on the learned interval, and linearly updating intervals, which are more flexible due to the inclusion of *prior-data conflicts* and their self-conjugacy.

5.1 Learning PAC intervals

We use the standard method of maximum a-posteriori probability (MAP) estimation to infer point estimates of probabilities, see Appendix A for details. These point estimates can then easily be turned into *probably approximately correct* (PAC) intervals via Hoeffding's inequality. Given $N = \#(s, a)$ samples and a fixed *error rate* γ, we use Hoeffding's inequality [Hoeffding, 1963] to compute the interval size $\delta = \sqrt{\log(2/\gamma)/2N}$. Using this δ, we then construct the intervals

$$\mathcal{P}(s, a, s_i) = [\max(\varepsilon, \tilde{P}(s, a, s_i) - \delta), \min(\tilde{P}(s, a, s_i) + \delta, 1)], \tag{1}$$

where $\tilde{P}$ is the (MAP) point estimate, and ε is again a small value to ensure that the interval lower bounds are non-zero. As a result, we have the following Proposition, the proof is a direct application of Hoeffding's inequality.

Proposition 1. *The true probability $P(s, a, s_i)$ lies within the learned interval $\mathcal{P}(s, a, s_i)$ with probability greater or equal to $1 - \gamma$.*

5.2 Learning Linearly Updating Probability Intervals

We use the Bayesian approach of *intervals with linearly updating conjugate priors* [Walter and Augustin, 2009] to learn intervals of probabilities. Each uncertain transition $\mathcal{P}(s, a, s_i)$ is assigned

a *prior interval* $\mathcal{P}_i = [\underline{\mathcal{P}}_i, \overline{\mathcal{P}}_i]$, and a *prior strength interval* $[\underline{n}_i, \overline{n}_i]$ that represents a minimum and maximum number of samples the prior interval is based on. The greater the values of the strength interval, the more emphasis is placed on the prior, and the more data is needed to significantly change the prior when computing the posterior. The greater the difference between the $\underline{n}_i$ and $\overline{n}_i$, the greater the difference between a *prior-data conflict* and a *prior-data agreement*.

Definition 3 (Posterior interval computation)**.** *The interval* $[\underline{\mathcal{P}}_i, \overline{\mathcal{P}}_i]$ *can be updated to* $[\underline{\mathcal{P}}'_i, \overline{\mathcal{P}}'_i]$*, using* $N = \#(s, a)$ *and* $k_i = \#(s, a, s_i)$*, as follows:*

$$\underline{\mathcal{P}}'_i = \begin{cases} \frac{\overline{n}_i \underline{\mathcal{P}}_i + k_i}{\overline{n}_i + N} & \text{if } {}^{k_i}/_{N} \geq \underline{\mathcal{P}}_i \text{ (prior-data agreement),} \\ \frac{\underline{n}_i \underline{\mathcal{P}}_i + k_i}{\underline{n}_i + N} & \text{otherwise (prior-data conflict).} \end{cases} \tag{2}$$

$$\overline{\mathcal{P}}'_i = \begin{cases} \frac{\overline{n}_i \overline{\mathcal{P}}_i + k_i}{\overline{n}_i + N} & \text{if } {}^{k_i}/_{N} \leq \overline{\mathcal{P}}_i \text{ (prior-data agreement),} \\ \frac{\underline{n}_i \overline{\mathcal{P}}_i + k_i}{\underline{n}_i + N} & \text{otherwise (prior-data conflict).} \end{cases} \tag{3}$$

The strength interval is updated by adding the number of samples N *to it:* $[\underline{n}'_i, \overline{n}'_i] = [\underline{n}_i + N, \overline{n}_i + N]$.

Key properties of linearly updating probability intervals.

- *Convergence in the infinite run.* Under the assumption that the true MDP does not change, each interval will converge to the exact transition probability when the total number of samples processed tends to infinity, regardless of how many samples are processed *per iteration* [Walter and Augustin, 2009]. This assumption is, however, not required for our work. If the true MDP changes over time, or is *adversarial* (i.e., a uMDP), our method is still applicable, but will not converge to a fixed MDP.
- *Prior-data conflict.* When the estimated probability ${}^{k_i}/_{N}$ lies outside the current interval, a so-called *prior data conflict* occurs. Consequently, if at some point we derive an interval that does not contain the true transition probability, the method will correct itself in later iterations.
- *Rate of convergence.* The strength of the prior $[\underline{n}_i, \overline{n}_i]$ controls the rate of convergence. When the data agrees with the prior, and the number of samples is equal to the upper bound of the prior strength, i.e. $\overline{n}_i = N$, the interval will halve in size. Due to the (linearly) increasing prior strength, prior-data conflicts arising from bad point estimates due to a low sample size have a declining effect over time, which means that, over multiple iterations, the intervals still converge towards the true probability, even if the individual point estimates never get close.

The initial values for both the prior intervals $[\underline{\mathcal{P}}_i, \overline{\mathcal{P}}_i]$ and the prior strengths $[\underline{n}_i, \overline{n}_i]$ of each state-action pair can be chosen freely.

A key requirement for computing robust policies on uMDPs is that the lower bound of every interval is strictly greater than zero (see Definition 2). If we assume that the intervals for the initial prior have a lower bound strictly greater than zero, the learning method will always update an interval to a new interval that has a lower bound greater than zero, yielding a valid uMDP. This closure property is formally stated in the following Proposition. The proof is relegated to Appendix B.

Proposition 2 (Closure under learning)**.** *Any new uncertain MDP* $\mathcal{M}'$ *computed according to Definition 3 using a prior uncertain MDP* $\mathcal{M}$ *with lower bounds strictly greater than zero will again have lower bounds greater than zero.*

Checking validity of the intervals. Since we learn individual intervals for transitions, it is not guaranteed that all intervals at a given state-action pair together form valid probability distributions. To that end, we check whether the sum of the lower bounds of the intervals is below one, and the sum of the upper bounds is above one, up to a certain tolerance ξ (we use `1e-8`). If either one of these conditions is not satisfied, we decrease (increase) the lower (upper) bounds by scaling them with a factor ${}^{(1-\xi)}/_{(\sum_i \underline{\mathcal{P}}_i)}$ (or ${}^{(1+\xi)}/_{(\sum_i \overline{\mathcal{P}}_i)}$ for the upper bound). That way, we guarantee that the intervals may form valid probability distributions. After computing a robust policy, we discard the adjusted intervals and continue learning with the intervals the learning process yielded previously.

5.3 Efficient exploration

Above, we assumed that a set of trajectories was given. To actually obtain the trajectories, we use the well-established *optimism in the face of uncertainty* principle. We compute the optimal policy for the optimistic extension (see Section 3) of the specification φ, i.e. φ_O, in the current uMDP and use this policy for exploration. To make exploration specification-driven, we only sample transitions along trajectories towards the target state(s) of the specification. When the last seen state has probability zero or one to reach the target for reachability, or reward zero or infinity, we restart. States satisfying these conditions can be found by analyzing the graph of the true MDP [Baier and Katoen, 2008].

Trajectories and iterations. As explained in Section 4, our method is iterative in terms of updating the uMDP $\mathcal{M}$ and computing a robust policy. After updating the uMDP, we also compute a new optimistic or reward-based exploration policy, as those depend on the intermediate uMDP. Each iteration consists of processing at least one, but possibly more trajectories. To determine how many trajectories to collect, we use a doubling-counting scheme, where we keep track of how often every state-action pair and transition is visited during exploration [Jin et al., 2020]. An iteration is completed when any of the counters is doubled with respect to the previous iteration. A detailed description of this schedule is given in Appendix C.

6 Experimental Evaluation

We implement our approach, with both linearly updating intervals (LUI) and PAC intervals (PAC), in Java on top of the verification tool PRISM [Kwiatkowska et al., 2011], together with a variant of value iteration to compute robust policies for uMDPs with convex uncertainties [Wolff et al., 2012]. We compare our method to point estimates derived via MAP-estimation (MAP) and with uMDPs learned by the UCRL2 reinforcement learning algorithm [Jaksch et al., 2010] (UCRL2). We make small modifications to UCRL2 to make it more comparable to our setting. In particular, we use optimistic policies for exploration, but robust policies to compute the performance, in contrast to the standard UCRL2 setting which only uses optimistic policies, see Appendix E for further details.

Without knowledge about the true MDP apart from Assumption 1, we have to define an appropriate prior interval for every transition. We set $\varepsilon = \texttt{1e-4}$ as constant and define the prior uMDP with intervals $\mathcal{P}_i = [\varepsilon, 1-\varepsilon]$ and strength intervals $[\underline{n}_i, \overline{n}_i] = [5, 10]$ at every transition $\mathcal{P}(s, a, s_i)$, as in Figure 2c. For MAP we use a prior of $\alpha_i = 10$ for all i. The same prior is used for the point estimates of both PAC and UCRL2, together with an error rate of $\gamma = 0.01$.

Evaluation metrics. We consider three metrics to evaluate the four learning methods.

- *Performance.* How does the robust policy computed on the learned model perform on the true MDP? We evaluate the probability of satisfying the given specification $\mathbb{P}_\pi^M(\Diamond T)$ or expected reward $\mathrm{R}_\pi^M(\Diamond T)$ of the robust policy π computed after each update of the model.
- *Performance Estimation Error.* How well do we *expect* a robust policy to perform on the true MDP based on the performance on the intermediate learned uMDP? We compute the difference between the performance of the robust policy on the learned uMDP (the worst-case performance) and the performance on the true MDP. While values closer to zero are preferable, we do not accept methods with positive estimation errors, since this indicates their estimated performance is not a lower (conservative) bound on the actual performance of the policy. In particular, an Estimation Error above zero shows the policy is misleading in terms of predicting its performance.
- *Model Error.* How can we measure errors in the estimate of the true MDP? For each transition, we compute the maximum distance between the true probability and the lower and upper bounds of the interval in the uMDP (or the point estimate for MAP-estimation), and then take the average over all these distances.

We benchmark our method using several well-known environments: the Chain Problem [Araya-López et al., 2011], Aircraft Collision Avoidance [Kochenderfer, 2015], a slippery Grid World [Derman et al., 2019], a 99-armed Bandit [Lattimore and Szepesvári, 2020], and two version of a Betting Game [Bäuerle and Ott, 2011]. For details on all these environments we refer to Appendix D. We

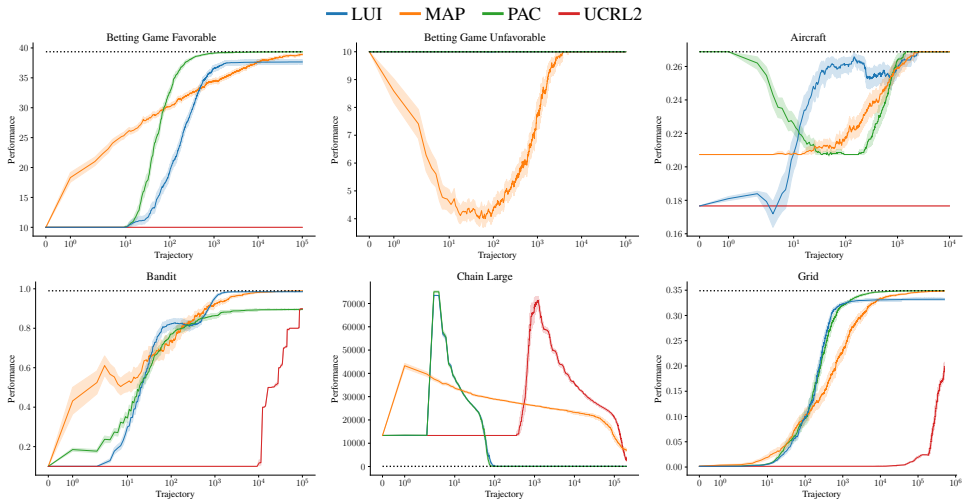


Figure 3: Comparison of the performance of robust policies on different environments against the number of trajectories processed (on log-scale). The dashed line indicates the optimal performance.

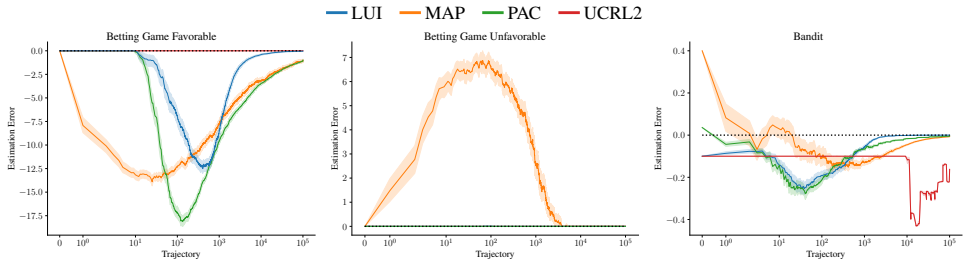


Figure 4: Estimation Error on the two Betting Game environments and the Bandit, against the number of trajectories processed (on log-scale).

highlight the Betting Game and Chain Problem environments here, as they will be used to explain some of the key observations we make from our experimental results.

- *Betting Game.* The agent starts with 10 coins and attempts to maximize the number of coins after six bets. When a bet is won, the number of coins placed is doubled; when lost, the number of coins placed is removed. The agent may bet 0, 1, 2, 5, or 10 coins. We consider two versions of the game, one which is favorable to the player, with a win probability of 0.8, and one that is unfavorable with a win probability of 0.2. After six bets the player receives a reward equal to the number of coins left. The specification is to maximize the reward.
- *Chain Problem.* We consider a chain problem Araya-López et al. [2011] with 30 states. There are three actions, one progresses with probability 0.8 to the next state, and resets the model to the initial state with probability 0.2. The second action does the same, but with reversed probabilities. The third action has probability 0.5 for both cases. Every action gets a reward of 1. As specification, we minimize the reward to reach the last state of the chain.

Results

We present an excerpt of our experimental results here, and refer the reader to Appendix F for the full set of results, which in particular also includes the Model Error metric and the Estimation Error for all environments. All experiments were performed on a machine with a 4GHz Intel Core i9 CPU, using a single core. Each experiment is repeated 100 times, and is reported with a 95% confidence interval.

Figure 3 shows the Performance of the robust policies computed via each learning method against the number of trajectories processed on the different environments. We first note that the Performance for LUI and PAC is roughly equivalent.

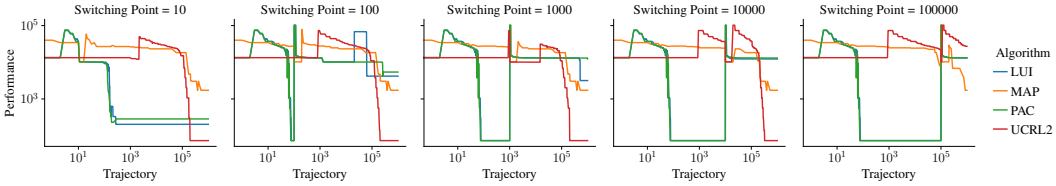


Figure 5: Environment change on the Chain Problem at different points ($10^1, \ldots, 10^5$).

UCRL2 is the slowest to converge to an optimal policy. This due to UCRL2 being a reinforcement learning algorithm, and thus it is slower in reducing the intervals in favor of a broader exploration. Interestingly, sometimes LUI gets stuck in a sub-optimal policy where PAC is able to find an optimal policy, and also the other way around, see e.g. the results for the Bandit and the Grid environments.

On the Chain environment, we see LUI and PAC (around trajectory 5), and UCRL2 (around trajectory 10^3) choose the wrong action(s), with an increase in Performance as a result (recall, the Chain Problem is a minimization problem). While all three methods manage to recover and then find the optimal policy, UCRL2 takes significantly longer: only after trajectory 10^5, i.e., it needs about $100\,000$ trajectories to recover, where LUI and PAC only need about 100 trajectories.

MAP-estimation typically sits between LUI/PAC and UCRL2. It is less sensitive to mistakes like the one discussed above, but has wider confidence intervals in general, and is less reliable in providing a conservative bound on its performance, as will be discussed below. Furthermore, we see that in the unfavorable Betting Game, only MAP-estimation gives sub-optimal performance, due to bad estimates. It is able to recover from this, but needs almost $10\,000$ trajectories to do so. Due to the low win probability in this Betting Game, a robust policy on the uMDPs is by default an optimal policy for the true MDP, and we see that LUI, PAC, and UCRL2 do not change to a sub-optimal policy.

Robust policies are conservative. Consider Figure 4. We note the undesirable behavior of having an Estimation Error above zero, which means the performance of the policy on the learned model was higher than the performance of that policy on the true MDP. MAP-estimation is particularly susceptible to this, while all three uMDP methods yield policies that are conservative in general, though some exceptions exist as shown in the full results in Appendix F.

Change of environment. Finally, we investigate the behavior of the learning methods when after a fixed number of trajectories the probabilities of the true MDP change. Figure 5 shows the performance of the robust policy for each learning method on the Chain environment. After $10^1, \ldots, 10^5$ trajectories, we change the environment by swapping the transition probabilities. As a result, the new optimal policy has to use the opposite action from the previous optimal policy. We note that the more trajectories were processed, the more difficulty LUI and PAC have in finding the new optimal policy, in contrast to UCRL2. Since all methods use optimism in the face of uncertainty for exploration, the greater the intervals, the better the exploration. If the change of environment happens when the intervals are still large, the exploration will exploit this and the newly collected data helps in determining the transition probabilities of the new environment. Hence, there is a direct trade-off between the rate of convergence and adaptability to changing environments.

7 Conclusion and Future Work

We presented a new Bayesian method that learns uMDPs to approximate an MDP, either via linearly updating intervals, or PAC-intervals. Robust policies computed on learned uMDPs are shown to be conservative and reliable in predicting their performance when applied on the MDP that is being learned. For future work, we aim to improve the adaptability of our method to the case of changing environments, and to extend it to uncertain POMDPs [Suilen et al., 2020, Cubuktepe et al., 2021]. While we do not see any immediate negative societal impacts of our work, we acknowledge that potential misuse of our work cannot be ruled out due to the generality of MDPs.

References

Mauricio Araya-López, Olivier Buffet, Vincent Thomas, and François Charpillet. Active learning of MDP models. In *EWRL*, volume 7188 of *LNCS*, pages 42–53. Springer, 2011.

Pranav Ashok, Jan Kretínský, and Maximilian Weininger. PAC statistical model checking for markov decision processes and stochastic games. In *CAV (1)*, volume 11561 of *LNCS*, pages 497–519. Springer, 2019.

Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

Nicole Bäuerle and Jonathan Ott. Markov decision processes with average-value-at-risk criteria. *Math. Methods Oper. Res.*, 74(3):361–379, 2011.

Dimitri P. Bertsekas. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005.

Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

Jacob Buckman, Carles Gelada, and Marc G Bellemare. The Importance of Pessimism in Fixed-Dataset Policy Optimization. In *ICLR*, pages 1–11. OpenReview.net, 2021.

Murat Cubuktepe, Nils Jansen, Sebastian Junges, Ahmadreza Marandi, Marnix Suilen, and Ufuk Topcu. Robust finite-state controllers for uncertain pomdps. In *AAAI*, pages 11792–11800. AAAI Press, 2021.

Esther Derman, Daniel J. Mankowitz, Timothy A. Mann, and Shie Mannor. A bayesian approach to robust reinforcement learning. In *UAI*, volume 115 of *Proceedings of Machine Learning Research*, pages 648–658. AUAI Press, 2019.

Ronan Fruit, Matteo Pirotta, Alessandro Lazaric, and Emma Brunskill. Regret minimization in mdps with options without prior knowledge. In *NIPS*, pages 3166–3176, 2017.

Vineet Goyal and Julien Grand-Clement. Robust markov decision process: Beyond rectangularity, 2020.

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

Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Mach. Learn.*, 110(3):457–506, 2021.

Bart Jacobs. A channel-based perspective on conjugate priors. *Math. Struct. Comput. Sci.*, 30(1): 44–61, 2020.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *J. Mach. Learn. Res.*, 11:1563–1600, 2010.

Chi Jin, Tiancheng Jin, Haipeng Luo, Suvrit Sra, and Tiancheng Yu. Learning adversarial markov decision processes with bandit feedback and unknown transition. In *ICML*, pages 4860–4869. PMLR, 2020.

Ying Jin, Zhuoran Yang, and Zhaoran Wang. Is Pessimism Provably Efficient for Offline RL? In *ICML*, pages 5084–5096. PMLR, 2021.

Mykel J Kochenderfer. *Decision making under uncertainty: theory and application*. MIT press, 2015.

Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch Reinforcement Learning. In Marco A Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State-of-the-Art*, pages 45–73. Springer Berlin Heidelberg, Berlin, Germany, 2012.

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

Shiau Hong Lim, Huan Xu, and Shie Mannor. Reinforcement learning in robust markov decision processes. In *NIPS*, pages 701–709, 2013.

Shie Mannor, Duncan Simester, Peng Sun, and John N. Tsitsiklis. Bias and variance approximation in value function estimates. *Manag. Sci.*, 53(2):308–322, 2007.

Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *CoRR*, abs/2006.16712, 2020.

Jun Morimoto and Kenji Doya. Robust reinforcement learning. *Neural Comput.*, 17(2):335–359, 2005.

Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Oper. Res.*, 53(5):780–798, 2005.

Amir Pnueli. The Temporal Logic of Programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977. doi: 10.1109/SFCS.1977.32.

Alberto Puggelli, Wenchao Li, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Polynomial-time verification of PCTL properties of mdps with convex uncertainties. In *CAV*, volume 8044 of *LNCS*, pages 527–542. Springer, 2013.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.

Paria Rashidinejad, Banghua Zhu, Cong Ma, Jiantao Jiao, and Stuart Russell. Bridging Offline Reinforcement Learning and Imitation Learning: A Tale of Pessimism. In *NeurIPS*. Curran Associates, Inc., 2021.

Reazul Hasan Russel and Marek Petrik. Beyond confidence regions: Tight bayesian ambiguity sets for robust mdps. In *NeurIPS*, pages 7047–7056, 2019.

Alexander L. Strehl and Michael L. Littman. An analysis of model-based interval estimation for markov decision processes. *J. Comput. Syst. Sci.*, 74(8):1309–1331, 2008.

Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite mdps: PAC analysis. *J. Mach. Learn. Res.*, 10:2413–2444, 2009.

Marnix Suilen, Nils Jansen, Murat Cubuktepe, and Ufuk Topcu. Robust policy synthesis for uncertain pomdps via convex optimization. In *IJCAI*, pages 4113–4120. ijcai.org, 2020.

Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. L^*-based learning of markov decision processes. In *FM*, volume 11800 of *LNCS*, pages 651–669. Springer, 2019.

Frits W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017.

Gero Walter and Thomas Augustin. Imprecision and prior-data conflict in generalized Bayesian inference. *Journal of Statistical Theory and Practice*, 3(1):255–271, 2009.

Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. Robust markov decision processes. *Math. Oper. Res.*, 38(1):153–183, 2013.

Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *CDC*, pages 3372–3379. IEEE, 2012.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We briefly reflect on this in the conclusion.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] All experiments were ran on a standard desktop PC, see Section 6.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Assumption 1in particular.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix B in particular.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material for the code and instructions on how to reproduce the results in the paper.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All priors and other parameters are mentioned where relevant.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All experiments are repeated 100 times and reported with 95% confidence intervals, as also mentioned in Section 6.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 6.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Appendix D for descriptions
   (b) Did you mention the license of the assets? [Yes] License is included in the supplementary material (GNU general public license).
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See the `models` folder in the supplementary material. In particular, we translated the existing models from the literature into the Prism language.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A MAP-estimation

We first describe a general setup for learning point estimates of probabilities via *maximum a-posteriori estimation* (MAP-estimation), and then how to transform these point estimates into intervals. For a fixed state-action pair (s, a) with m different successor states $s_1, \ldots, s_m$ that is sampled $N = \#(s, a)$ times, and where each successor state is observed $k_i = \#(s, a, s_i)$ times for $i = 1, \ldots, m$, we have the *multinomial likelihood*

$$Mn(k_1, \ldots, k_n \mid P(s, a, \cdot)) = \frac{N!}{k_1! \cdot \ldots \cdot k_m!} \cdot \prod_{i=1}^{m} P(s, a, s_i)^{k_i}, \tag{4}$$

where the transition probabilities $P(s, a, s_i)$ are unknown and given by a Dirichlet distribution with parameters $\alpha_1, \ldots, \alpha_m$ [Bishop, 2007]:

$$Dir(P(s, a, \cdot) \mid \alpha_1, \ldots, \alpha_m) \propto \prod_{i=1}^{m} P(s, a, s_i)^{\alpha_i - 1}. \tag{5}$$

The Dirichlet distribution is a *conjugate prior* to the multinomial likelihood, meaning that we do not need to compute the posterior distribution explicitly, but instead we just update the parameters of the prior Dirichlet distribution. Formally, conjugacy is a closure property, see [Jacobs, 2020].

Given a prior Dirichlet distribution with parameters $\alpha_1, \ldots, \alpha_m$, and observing the i-th successor state k_i times, the posterior Dirichlet distribution is given by simply adding k_i to parameter α_i:

$$Dir(P(s, a, \cdot) \mid \alpha_1 + k_1, \ldots, \alpha_m + k_m). \tag{6}$$

Having computed the posterior Dirichlet distribution, MAP-estimation can be used to infer the probabilities. These estimates are given by the *mode* of each parameter of the posterior distribution:

$$\tilde{P}(s, a, s_i) = \frac{\alpha_i - 1}{(\sum_{j=1}^{m} \alpha_j) - m}. \tag{7}$$

When all parameters α_i are equal, MAP-estimation yields uniform distributions.

B Proof Proposition 1.

The proof of Proposition 1 is straightforward and relies on the observation that for any amount of finite data, a single update can only grow closer to zero but not become zero, and that iterated updates converge to the true probability which is assumed to be non-zero.

Proof. For any transition (s, a, s_i) the following holds. Assume, for the prior uMDP $\mathcal{M}$, all lower bounds are strictly greater than 0. By Asm. 1, all transitions with intervals have a probability $0 < p < 1$ in the true MDP. Each iteration we sample a finite amount of data, yielding a point estimate $k_i/N \geq 0$. Then both computations in Eqn. (2) will again result in a new lower bound strictly greater than zero for posterior uMDP $\mathcal{M}'$. Over multiple iterations and in the infinite run, the interval converges to the single element set of the true probability p, for which we know $p \neq 0$ by assumption. $\square$

C Exploration Scheme

We detail the exploration scheme in Algorithm 1 below.

Algorithm 1 Exploration scheme.

Input: K : number of trajectories.
Input: H : max trajectory length.
Input: $M(S, s_I, A, P, R)$: underlying MDP.
Input: φ : specification.
Input: $\mathcal{A}$: algorithm for exploration and robust verification.

```
for s, a ∈ S × A do
    ▷ Initialize counters
    #(s, a) = 0
    #(s, a, s') = 0 : ∀s' ∈ S
    #_i(s, a) = 0
    #_i(s, a, s') = 0 : ∀s' ∈ S
end for
for k ∈ [1, ··· , K] do
    if ∃s, a ∈ S × A : #_i(s, a) >= #(s, a) then
        ▷ Compute new policies.
        Give iteration counters #_i(s, a) and #_i(s, a, s') to A
        Get sampling policy from A : π_sampling
        Get robust policy from A : π_robust
        Evaluate π_robust on M according to φ
        ▷ Update global counters
        #(s, a) += #_i(s, a) : ∀s, a ∈ S × A
        #(s, a, s') += #_i(s, a, s') : ∀s, a ∈ S × A × S
        ▷ Reset iteration counters
        #_i(s, a) = 0 : ∀s, a ∈ S × A
        #_i(s, a, s') = 0 : ∀s, a ∈ S × A × S
    end if
    Sample τ from M following π_sampling with max size H
    Increment #_i(s, a) and #_i(s, a, s') according to τ.
end for
```

D Detailed Problem Domain Descriptions

We give a detailed description of each model we use in the experimental evaluation below, together with the specification and (if applicable) its source.

Chain problem. We consider a larger version of the chain problem Araya-López et al. [2011] with 30-states. There are three actions, one progresses with probability 0.8 to the next state, and resets the model to the initial state with probability 0.2. The second action does the same, but with reversed probabilities. The third action has probability 0.5 for both progressing and resetting. Every action gets a reward of 1. As specification, we minimize the reward to reach the last state of the chain: $\mathrm{R}_{\mathrm{Min}}(\Diamond T)$. This can be seen as an instance of the stochastic shortest path problem [Bertsekas, 2005].

Aircraft collision avoidance. We model a small, simplified instance of the aircraft collision avoidance problem Kochenderfer [2015]. Two aircraft, one controlled, one adversarial, approach each other. The controlled aircraft can increase, decrease, or stay at the current altitude with a success probability of 0.8. The adversarial aircraft can do the same, but does so with probabilities 0.3, 0.3, 0.4 respectively. To goal is to maximize the probability of the two aircraft passing each other without a collision, expressed by the temporal logic specification $\mathbb{P}_{\mathrm{Max}}(\neg\text{collision} \,\mathsf{U}\, \text{passed})$.

Grid world. We consider a slippery 10×10 grid world as in Derman et al. [2019], where a robot starts in the north-west corner and has to navigate towards a target position. The robot can move in each of the four cardinal directions with a success probability of 0.55, and a probability of 0.15 to move in each of the other directions. Throughout the grid, the robot has to avoid traps. The model

has 100 states and 1450 transitions. We consider *safety* specifications that maximize the probability for reaching the north-east (NE) corner without getting trapped, i.e. $\mathbb{P}_{\text{Max}}(\neg\text{trapped} \,\mathsf{U}\, \text{NE})$.

Bandit. We consider a 99-armed bandit Lattimore and Szepesvári [2020], where each arm (action) has an increased probability of success, $0.01, 0.02, \ldots, 0.99$ respectively. The goal is to find the action that has the highest probability of success: $\mathbb{P}_{\text{Max}}(\Diamond \text{ success})$.

Betting game. We consider a betting game from [Bäuerle and Ott, 2011], in which an agent starts with 10 coins and attempts to maximize the number of coins after 6 bets. When a bet is won, the amount of coins placed is doubled, when lost the amount of coins placed is removed. The agent may place bets of 0, 1, 2, 5, and 10 coins. We consider two versions of the game, one which is favorable to the player, with a win probability of 0.8, and one that is unfavorable to the player, with a win probability of 0.2. After six bets the player receives a state-based reward that equals the number of coins left. This yields an MDP of 300 states and 1502 transitions. The specification is to maximize the reward after for reaching a terminal state after these six bets.

E Comparison with UCRL2.

UCRL2 [Jaksch et al., 2010] is a reinforcement learning algorithm, meaning it is designed to collect reward while exploring, in contrast to our setting where we have a clear separation between exploration and policy computation (i.e. computing the reward). We use UCRL2 to learn a uMDP, but then compute a robust policy on that learned model, instead of an optimistic policy as is standard for UCRL2. Furthermore, we make the following changes to the UCRL2 algorithm to make it more comparable to our setting:

1. Just as with the PAC intervals, we use MAP-estimation instead of the frequentist approach to estimate the point estimates.
2. UCRL2 does not compute individual intervals, but a set of distributions around the distribution of point estimates. In particular, it takes the set of distributions for which the ℓ^1 norm to the estimated distribution is less than or equal to

$$\sqrt{\frac{14|S|\log(2|A|t_k \cdot {}^1\!/\!\gamma)}{\max(1, N)}}, \tag{8}$$

   see [Jaksch et al., 2010] for details on this bound. We use this bound for the individual intervals (bounded by ε to ensure lower bounds greater than zero). As the robust value iteration for uMDPs restricts to valid distributions with the product of these intervals, we do not use the ℓ^1 norm, but put in the intervals derived from MAP-estimation with this bound.
3. Finally, the UCRL2 algorithm assumes that the reward function is also unknown, and attempts to learn that as well. We assume the reward function to be known in our setting.

F Complete Experimental Evaluation

In Figure 6 we show the Performance, Model Error, and Estimation Error for all six environments and all four learning methods. Figure 7 shows the results for switching the probabilities on the Chain environment, ordered by trajectory in which the switch occurs and by learning algorithm. For our conclusions from these experiments we refer the reader back to Section 6 in the main text.

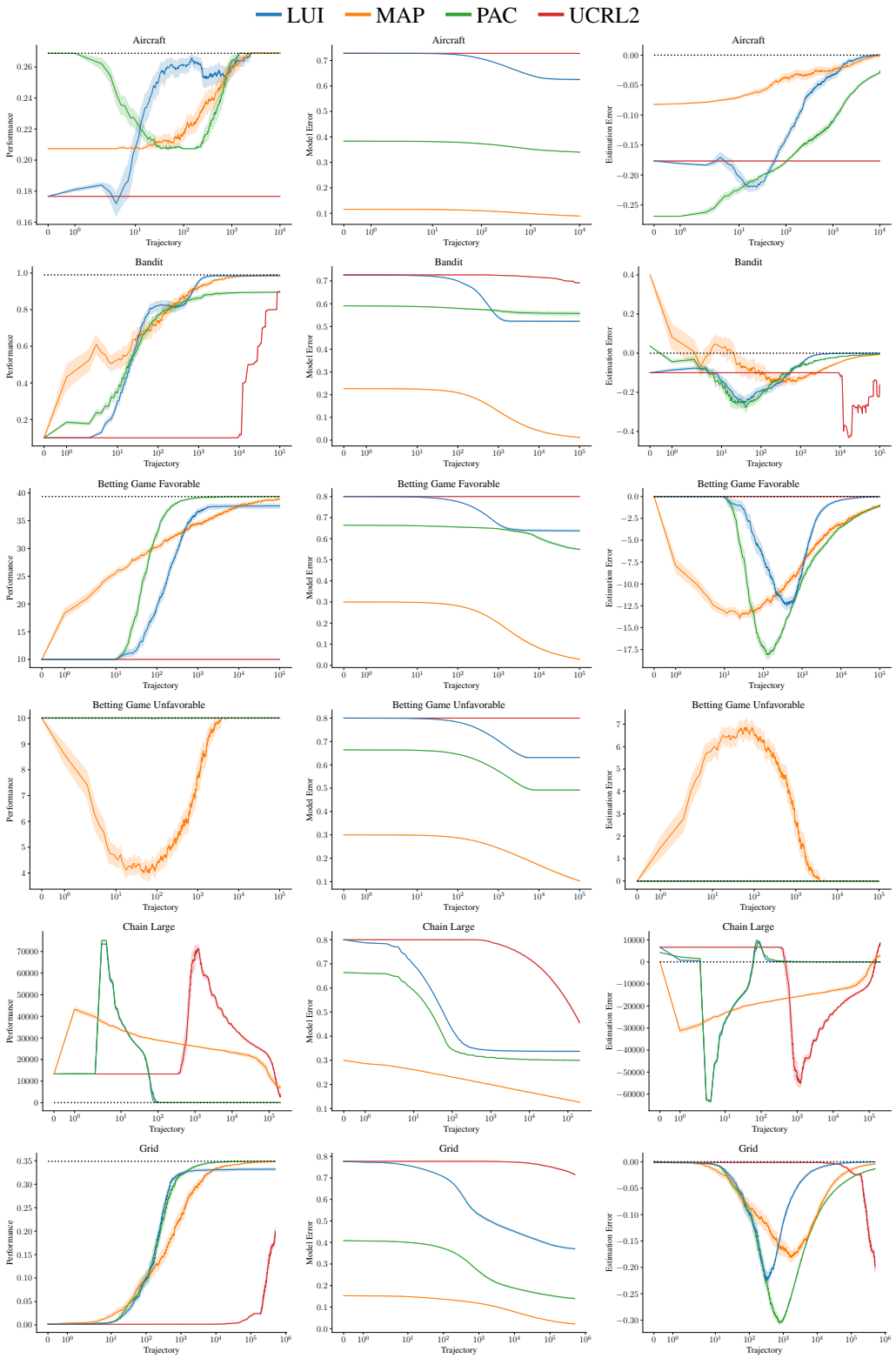


Figure 6: Performance, Model Error, and Estimation Error for all four learning methods on all six environments.

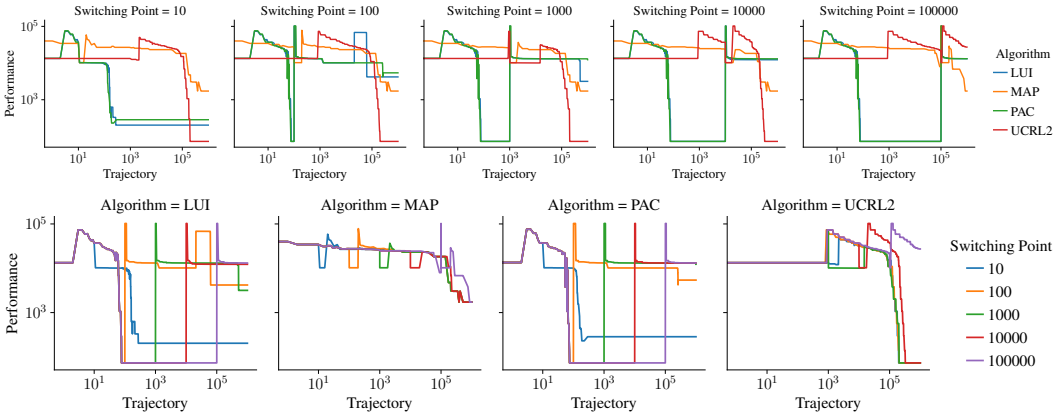


Figure 7: Switching the true probabilities in the Chain environment after $10^1, \ldots, 10^5$ trajectories, ordered by switching point (top) and by learning method (bottom).