Counterexample-Guided Strategy Improvement for POMDPs Using Recurrent Neural Networks

Steven Carr[1]*, **Nils Jansen**[2]*, **Ralf Wimmer**[3,4], **Alexandru Serban**[2], **Bernd Becker**[3] and **Ufuk Topcu**[1]

[1] The University of Texas at Austin
[2] Radboud University, Nijmegen, The Netherlands
[3] Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany
[4] Concept Engineering GmbH, Freiburg im Breisgau, Germany
stevencarr@utexas.edu, n.jansen@science.ru.nl

Abstract

We study strategy synthesis for partially observable Markov decision processes (POMDPs). The particular problem is to determine strategies that provably adhere to (probabilistic) temporal logic constraints. This problem is computationally intractable and theoretically hard. We propose a novel method that combines techniques from machine learning and formal verification. First, we train a recurrent neural network (RNN) to encode POMDP strategies. The RNN accounts for memory-based decisions without the need to expand the full belief space of a POMDP. Secondly, we restrict the RNN-based strategy to represent a finite-memory strategy and implement it on a specific POMDP. For the resulting finite Markov chain, efficient formal verification techniques provide provable guarantees against temporal logic specifications. If the specification is not satisfied, counterexamples supply diagnostic information. We use this information to improve the strategy by iteratively training the RNN. Numerical experiments show that the proposed method elevates the state of the art in POMDP solving by up to three orders of magnitude in terms of solving times and model sizes.

1 Introduction

Autonomous agents that make decisions under uncertainty and incomplete information can be mathematically represented as partially observable Markov decision processes (POMDPs). In this setting, while an agent makes decisions within an environment, it obtains *observations* and infers the likelihood of the system being in a certain state, known as the *belief* state. POMDPs are effective in modeling a number of real-world applications, see for instance [Kaelbling *et al.*, 1998]. Traditional POMDP problems typically seek to compute a strategy that maximizes a cumulative reward over a finite horizon.

However, the agent's behavior is often required to obey more complicated specifications. For example, reachability, liveness or, more generally, specifications expressed in temporal logic (e. g. LTL [Pnueli, 1977]) describe tasks that cannot be expressed using reward functions [Littman *et al.*, 2017].

Strategy synthesis for POMDPs is a difficult problem, both from the theoretical and the practical perspective. For infinite- or indefinite-horizon problems, computing an optimal strategy is undecidable [Madani *et al.*, 1999]. Optimal action choices depend on the whole history of observations and actions, and thus require an infinite amount of memory. When restricting the specifications to maximizing accumulated rewards over a finite horizon and also limiting the available memory, computing an optimal strategy is PSPACE-complete [Papadimitriou and Tsitsiklis, 1987]. This problem is, practically, intractable even for small instances [Meuleau *et al.*, 1999]. Moreover, even when strategies are restricted to be *memoryless*, finding an optimal strategy within this set is still NP-hard [Vlassis *et al.*, 2012]. For more general specifications like LTL properties, synthesis of strategies with limited memory is even harder, namely EXPTIME-complete [Chatterjee *et al.*, 2015]).

The intractable nature of finding exact solutions in these problems gave rise to approximate [Hauskrecht, 2000], point-based [Pineau *et al.*, 2003], or Monte-Carlo-based [Silver and Veness, 2010] methods. However, none of these approaches provides guarantees for temporal logic specifications. [Chatterjee *et al.*, 2015] studies such problems on a theoretical level. The tool PRISM-POMDP [Norman *et al.*, 2017] actually provides guarantees by approximating the belief space into a fully observable belief MDP, but is restricted to small examples. Other techniques, such as those employing an incremental satisfiability modulo theory (SMT) solver over a bounded belief space [Wang *et al.*, 2018] or a simulation over sets of belief models [Haesaert *et al.*, 2018], are also restricted to small examples. Finally, [Junges *et al.*, 2018] constructs finite-state controllers for POMDPs using parameter synthesis for Markov chains [Hahn *et al.*, 2010; Junges *et al.*, 2019] by applying convex optimization techniques [Cubuktepe *et al.*, 2017; 2018]. Their procedure involves constructing a product of the POMDP and an automaton for temporal logic constraints, which can cause a substantial blow-up in the state space.

Although strategy synthesis for POMDPs is hard, an available *candidate strategy* resolves the nondeterminism and par-

*Contact Authors

tial observability for a POMDP and yields a so-called induced discrete-time Markov chain (MC). For this simpler model, verification methods are capable of efficiently certifying temporal logic constraints and reward specifications for billions of states [Baier and Katoen, 2008]. Tool support is available via probabilistic model checkers such as PRISM [Kwiatkowska *et al.*, 2011] or Storm [Dehnert *et al.*, 2017].

There remains a dichotomy between directly synthesizing an optimal strategy and the efficient verification of a candidate strategy. The key questions are (1) how to generate a "good" strategy in the first place and (2) how to improve a strategy if verification refutes the specification. Machine learning and formal verification techniques address these questions separately. In this paper, we combine methods from both fields in order to guarantee that a candidate strategy learned through machine learning provably satisfies temporal logic specifications.

At first, we learn a randomized strategy* via recurrent neural networks (RNNs) [Hochreiter and Schmidhuber, 1997] and data stemming from knowledge of the underlying structure of POMDPs. We refer to the resulting trained RNN as the *strategy network*. RNNs are a good candidate for learning a strategy because they can successfully represent temporal dynamic behavior [Pascanu *et al.*, 2013].

Secondly, we extract a concrete (memoryless randomized) candidate strategy from the RNN and use it directly on a given POMDP, resulting in the MC induced by the POMDP and the strategy. Formal verification reveals whether specifications are satisfied or not. In the latter case, we generate a so-called counterexample [Wimmer *et al.*, 2014], which points to parts of the MC (and by extension of the POMDP) that are critical for the specification. For those critical parts, we use a linear programming (LP) approach that locally improves strategy choices (without any guarantees on the global behavior). From the improved strategy, we generate new data to retrain the RNN. We iterate that procedure until the strategy network yields satisfactory results.

While the strategies are memoryless, allowing for randomization over possible choices – relaxing determinism – is often sufficient to capture necessary variability in decision-making. The intuition is that *deterministic* choices at a certain state may need to vary depending on previous decisions, thereby trading off memory. However, randomization in combination with finite memory may supersede infinite memory for many cases [Amato *et al.*, 2010; Junges *et al.*, 2018]. We encode finite memory directly into a POMDP by extending its state space. We can then directly apply our method to create *finite-state controllers (FSCs)* [Meuleau *et al.*, 1999].

As previously discussed, the investigated problem is undecidable for POMDPs [Madani *et al.*, 1999] and therefore the approach is naturally incomplete. Soundness is provided, as verification yields hard guarantees on the quality of a strategy.

Related Work

Besides the publications mentioned earlier, we list several results employing RNN architectures for POMDPs, all of them within the policy gradient class of algorithms specific to reinforcement learning [Sutton *et al.*, 2000]. In this setting, the strategy is parameterized and updated by performing gradient ascent on the error function (typically chosen to maximize the discounted reward).

In order to cope with arbitrary memory in POMDPs, policy gradient methods need some notion of memory. RNNs are suitable for this task because (1) they are differentiable end-to-end and (2) they are designed to exhibit dynamic temporal behavior. [Wierstra *et al.*, 2007] were the first to employ an RNN to learn (finite-memory) strategies for POMDPs, using an long short-term memory (LSTM) architecture which is able to leverage both long and short term events in the past.

Recent progress in deep learning enabled scaling neural networks (NNs) to solve complex problems. For example, [Mnih *et al.*, 2015] developed an NN - based Q-learning algorithm able to play video games straight from video frames, under partial observability. Instead of using RNNs, the memory problem is solved by replaying a series of frames at every step. Later, [Hausknecht and Stone, 2015] added an LSTM cell to enhance the algorithm's capacity with both long and short term memory. The field has rapidly moved to explore new ways of improving the memory representation [Parisotto and Salakhutdinov, 2018] [Pritzel *et al.*, 2017] [Santoro *et al.*, 2018]. However, even though they yield good performance on a variety of tasks, these methods do not provide any guarantees on the strategies learned.

2 Preliminaries

A *probability distribution* over a finite or countably infinite set X is a function $\mu : X \to [0,1] \subseteq \mathbb{R}$ with $\sum_{x \in X} \mu(x) = \mu(X) = 1$. The set of all distributions on X is $Distr(X)$. The support of a distribution μ is $\text{supp}(\mu) = \{x \in X \mid \mu(x) > 0\}$.

(PO)MDPs. A *Markov decision process (MDP) M* is a tuple $M = (S, Act, \mathscr{P})$ with a finite (or countably infinite) set S of *states*, a finite set Act of *actions*, and a *transition function* $\mathscr{P} \colon S \times Act \to Distr(S)$. We use a reward function $r \colon S \times a \to \mathbb{R}$. A finite *path* π of an MDP M is a sequence of states and actions; $\text{last}(\pi)$ is the last state of π. The set of finite paths of M is Paths^M_{fin}. A *discrete-time Markov chain (MC)* is an MDP with $|Act(s)| = 1$ for all $s \in S$. A *strategy* γ for an MDP M is a function $\gamma \colon \mathsf{Paths}^M_{fin} \to Distr(Act)$ with $\text{supp}\big(\gamma(\pi)\big) \subseteq Act\big(\text{last}(\pi)\big)$ for all $\pi \in \mathsf{Paths}^M_{fin}$. A strategy γ is *memoryless* if $\text{last}(\pi) = \text{last}(\pi')$ implies $\gamma(\pi) = \gamma(\pi')$ for all $\pi, \pi' \in \mathsf{Paths}^M_{fin}$.

Definition 1 (Induced Markov Chain) *For an MDP $M = (S, Act, \mathscr{P})$ and a strategy $\gamma \in \Gamma^M$, the MC induced by M and γ is given by $M^\gamma = (\mathsf{Paths}^M_{fin}, P^\gamma)$ where:*

$$P^\gamma(\pi, \pi') = \begin{cases} \mathscr{P}(\text{last}(\pi), a, s') \cdot \gamma(\pi)(a) & \text{if } \pi' = \pi a s', \\ 0 & \text{otherwise.} \end{cases}$$

Definition 2 (POMDP) *A* partially observable Markov decision process (POMDP) *is a tuple $\mathscr{M} = (M, Z, O)$, with $M = (S, Act, \mathscr{P})$ the* underlying MDP of *$\mathscr{M}$, Z a finite set of observations and $O \colon S \to Z$ the* observation function.

The set of all finite observation-action sequences for a POMDP $\mathscr{M}$ is denoted by $\mathsf{ObsSeq}^{\mathscr{M}}_{fin}$.

*Also referred to as stochastic strategy or policy.

Definition 3 (POMDP Strategy) *An* observation-based strategy *for a POMDP $\mathcal{M}$ is a function $\gamma\colon \mathsf{ObsSeq}^{\mathcal{M}}_{fin} \to Distr(Act)$ such that $\mathrm{supp}\big(\gamma(O(\pi))\big) \subseteq Act\big(\mathrm{last}(\pi)\big)$ for all $\pi \in \mathsf{Paths}^{M}_{fin}$. $\Gamma^{\mathcal{M}}_{z}$ is the set of observation-based strategies for $\mathcal{M}$.*

A *memoryless* observation-based strategy $\gamma \in \Gamma^{\mathcal{M}}_{z}$ is analogous to a memoryless MDP strategy, formally we simplify to $\gamma\colon Z \to Distr(Act)$, i. e., we decide based on the current observation only. Similarly, a POMDP together with a strategy yields an induced MC as in Def. 1, resolving all nondeterminism and partial observability. A general POMDP strategy can be represented by *infinite-state controllers*. Strategies are often restricted to finite memory; this amounts to using finite-state controllers (FSCs) [Meuleau *et al.*, 1999].

Definition 4 (FSC) *A k-FSC for a POMDP is a tuple $\mathcal{A} = (N, n_I, \gamma_\alpha, \delta)$ where N is a* finite set *of k memory nodes, $n_I \in N$ is the initial memory node, γ is the action mapping $\gamma_\alpha\colon N \times Z \to Distr(Act)$ and δ is the memory update $\delta\colon N \times Z \times Act \to N$. Let $\gamma_{\mathcal{A}} \in \Gamma^{\mathcal{M}}_{z}$ denote the observation-based strategy represented by the FSC $\mathcal{A}$.*

The product $\mathcal{M} \times \mathcal{A}$ of a POMDP and a k-FSC yields a (larger) "flat" POMDP where the memory update is directly encoded into the state space [Junges *et al.*, 2018]. The action mapping γ_α is left out of the product. A memoryless strategy $\gamma \in \Gamma^{\mathcal{M}\times\mathcal{A}}_{z}$ then determines the action mapping and can be projected to the finite-memory strategy $\gamma_{\mathcal{A}} \in \Gamma^{\mathcal{M}}_{z}$.

Specifications

We consider linear-time temporal logic (LTL) properties [Pnueli, 1977]. For a set of atomic propositions AP, which are either satisfied or violated by a state, and $a \in AP$, the set of LTL formulas is given by:

$$\Psi ::= a \mid (\Psi \wedge \Psi) \mid \neg\Psi \mid \bigcirc \Psi \mid \Box\Psi \mid (\Psi \,\mathsf{U}\, \Psi)\,.$$

Intuitively, a path π satisfies the proposition a if its first state does; $(\psi_1 \wedge \psi_2)$ is satisfied, if π satisfies both ψ_1 and ψ_2; $\neg\psi$ is true on π if ψ is not satisfied. The formula $\bigcirc\psi$ holds on π if the subpath starting at the second state of π satisfies ψ. The path π satisfies $\Box\psi$ if all suffixes of π satisfy ψ. Finally, π satisfies $(\psi_1 \,\mathsf{U}\, \psi_2)$ if there is a suffix of π that satisfies ψ_2 and all longer suffixes satisfy ψ_1. $\Diamond\psi$ abbreviates $(\mathrm{true}\,\mathsf{U}\,\psi)$.

For POMDPs, one wants to synthesize a strategy such that the probability of satisfying an LTL-property respects a given bound, denoted $\varphi = \mathbb{P}_{\sim\lambda}(\psi)$ for $\sim \in \{<, \leq, \geq, >\}$ and $\lambda \in [0,1]$. In addition, *undiscounted expected reward properties* $\varphi = \mathbb{E}_{\sim\lambda}(\Diamond a)$ require that the expected accumulated cost until reaching a state satisfying a respects $\lambda \in \mathbb{R}_{\geq 0}$.

If φ (either LTL or expected reward specification) is satisfied in a (PO)MDP $\mathcal{M}$ under γ, we write $\mathcal{M}^\gamma \models \varphi$, that is, the specification is satisfied in the induced MC, see Def. 1. While determining an appropriate strategy is still efficient for MDPs, this problem is in general undecidable for POMDPs [Chatterjee *et al.*, 2016]. In particular, for MDPs, to check the satisfaction of a general LTL specification one needs memory. Typically, tools like PRISM [Kwiatkowska *et al.*, 2011] compute the product of the MDP and a deterministic Rabin automaton. In this product, reachability of so-called accepting end-components ensures the satisfaction of the LTL property. This reachability probability can be determined in polynomial time. PRISM-POMDP [Norman *et al.*, 2017] handles the problem similarly for POMDPs, but note that a strategy needs memory not only for the LTL specification but also for observation dependencies.

Finally, given a (candidate) strategy γ, checking whether $\mathcal{M}^\gamma \models \varphi$ holds can be done both for MDPs and POMDPs in polynomial time. For more details we refer to [Baier and Katoen, 2008].

3 Synthesis Procedure

Formal problem statement. For a POMDP $\mathcal{M}$ and a specification φ, where either $\varphi = \mathbb{P}_{\sim\lambda}(\psi)$ with ψ an LTL formula, or $\varphi = \mathbb{E}_{\sim\lambda}(\Diamond a)$, the problem is to determine a (finite-memory) strategy $\gamma \in \Gamma^{\mathcal{M}}_{z}$ such that $\mathcal{M}^\gamma \models \varphi$.

If such a strategy does not exist, the problem is infeasible.

Outline

The workflow of the proposed approach is illustrated in Fig. 1: We *train* an RNN using observation-action sequences generated from an initial strategy as discussed in Sect. 3.1. The trained *strategy network* represents an observation-based strategy, taking as input an observation-action sequence and returning a distribution over actions, see Def 3. For a POMDP $\mathcal{M}$, we use the output of the strategy network in order to resolve nondeterminism. The strategy network is thereby used to extract a *memoryless strategy* $\gamma \in \Gamma^{\mathcal{M}}$ and as a result we obtain the induced MC $\mathcal{M}^\gamma$. *Model checking* of this induced MC evaluates whether the specification φ is satisfied or not for the extracted strategy. In the former case, the synthesis procedure is finished. The extraction and evaluation is explained in Sect. 3.2.

If the specification is not satisfied, we obtain a *counterexample* highlighting critical states of the POMDP. We employ

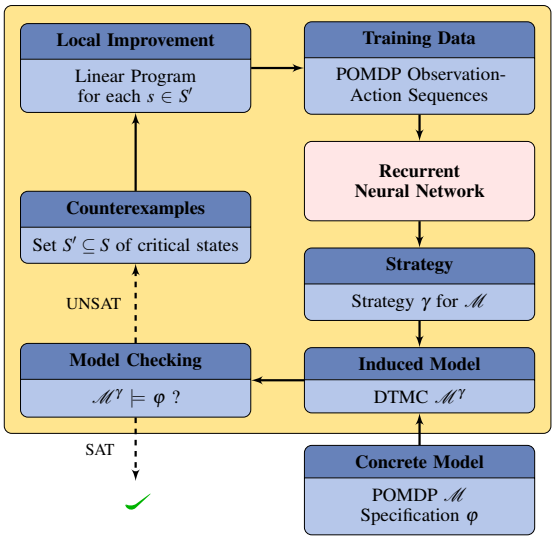


Figure 1: Flowchart of the RNN-based refinement loop

a linear programming (LP) approach that locally *improves* action choices of the current strategy at these critical states, see Sect. 3.3. Afterwards, we retrain the RNN by generating new observation-action sequences obtained from the new strategy. We iterate this procedure until the specification is satisfied or a fixed iteration threshold is reached. For cases where we need to further improve, we use domain knowledge to create a specific memory-update function of a k-FSC $\mathscr{A}$, see Def. 4. Then, we compute the product $\mathscr{M}' = \mathscr{M} \times \mathscr{A}$. We iterate our method with $\mathscr{M}'$ as starting point and thereby determine a concrete k-FSC including the action mapping.

3.1 Learning Strategies with RNNs

As mentioned in Section 1, policy gradient algorithms are used to map observations to actions and are not well suited for POMDPs due to their inability to cope with arbitrary memory. To overcome this weakness, we design our method to make explicit use of memory using RNNs - a family of neural networks designed to exhibit dynamic temporal behavior.

Constructing the strategy network. We use the long short-term memory (LSTM) architecture [Hochreiter and Schmidhuber, 1997] in a similar fashion to policy gradient methods and model the output as a probability distribution on the action space (described formally by $\hat{\gamma}\colon \mathsf{ObsSeq}^{\mathscr{M}}_{fin} \to \mathit{Distr}(\mathit{Act})$). By having stochastic output units, we avoid computing gradients on the internal belief states, see [Meuleau *et al.*, 1999] for a similar approach. Using back propagation through time, we can update the strategy during training. Thus, for a given observation-action sequence from $\mathsf{ObsSeq}^{\mathscr{M}}_{fin}$, the model learns a strategy $\hat{\gamma} \in \Gamma^{\mathscr{M}}_{z}$. The output is a discrete probability distribution over the actions Act, represented using a final softmax layer.

RNN training. We train the RNN using a slightly modified version of sampling re-usable trajectories [Kearns *et al.*, 2000]. In particular, for a POMDP $\mathscr{M} = (M, Z, O)$ and a specification φ, instead of randomly generating observation sequences, we first compute a strategy $\gamma \in \Gamma^M$ of the underlying MDP M that satisfies φ. Then we sample uniformly over all states of the MDP and generate finite paths (of a fixed maximal length) from $\mathsf{Paths}^{M^\gamma}_{fin}$ of the induced MC M^γ, thereby creating multiple trajectory trees. For each finite path $\pi \in \mathsf{Paths}^{M^\gamma}_{fin}$, we generate one possible observation-action sequence $\pi_z \in \mathsf{ObsSeq}^{\mathscr{M}}_{fin}$ such that $\pi = z_0, a_0, \ldots, a_{n-1}, z_n$ with $z_i = O(\pi[i])$, where $\pi[i]$ denotes the i-th state of π for all $1 \leq i \leq n$. We form the training set $\mathscr{D}$ from a (problem specific) number of m observation-action sequences with observations as input and actions as output labels. Both input and output sets were processed using one-hot-encoding. To fit the RNN model, we use the Adam optimizer [Kingma and Ba, 2014] with a cross-entropy error function.

Sampling large environments. In a POMDP $\mathscr{M}$ with a large state space ($|S| > 10^5$), computing the underlying MDP strategy $\gamma \in \Gamma^M$ affects the performance of the procedure. In such cases, we restrict the sampling to a smaller environment that shares the observation Z and action spaces Act with $\mathscr{M}$. For example, consider a gridworld scenario with a moving obstacle that has the same underlying probabilistic movement for different problem sizes; such a framework can provide a similar dataset regardless of the size of the grid.

3.2 Strategy Extraction and Evaluation

We first describe how to extract a memoryless strategy from the strategy network for a specific POMDP, then we formalize the extension to FSCs to account for finite memory. Afterwards, we shortly explain how to evaluate the resulting strategies.

Given a POMDP $\mathscr{M}$, we use the trained strategy network $\hat{\gamma}\colon \mathsf{ObsSeq}^{\mathscr{M}}_{fin} \to \mathit{Distr}(\mathit{Act})$ directly as observation-based strategy. Note that the RNN is inherently a predictor for the distribution over actions and will not always deliver the same output for one input. While we always use the first prediction we obtain, one may also sample several predictions and take the average of the output distributions.

Extension to FSCs

As mentioned before, LTL specifications as well as observation-dependencies in POMDPs require memory. Consider therefore a general FSC $\mathscr{A} = (N, n_I, \gamma, \delta)$ as in Def. 4. We first predefine the memory update function δ in a problem-specific way, for instance, δ changes the memory node when an observation is repeated. Consider observation sequence $\pi_z \in \mathsf{ObsSeq}^{\mathscr{M}}_{fin}$ with $\pi_z = z_0, a_0, \ldots, z_n$. Assume, the FSC is in memory node $n_k \in N$ at position i of π_z. We define $\delta(n_k, z_i, a_i) = n_{k+1}$, if $\pi_z[i] = (z_i, a_i)$, and there exists a $j < i$ such that $\pi_z[j] = (z_j, a_j)$ with $z_i = z_j$. Similarly, we account for specific memory choices akin to the relevant LTL specification.

Once δ has been defined, we compute a product POMDP $\mathscr{M} \times \mathscr{A}$ which creates a state space over $S \times N$. The training process is similar to the method outlined above but instead of generating observation-action sequences from $\mathsf{ObsSeq}^{\mathscr{M}}_{fin}$, we generate observation-node-action sequences $(z_0, n_0), a_0, \ldots, a_{n-1}, (z_n, n_n)$ from $\mathsf{ObsSeq}^{\mathscr{M}\times\mathscr{A}}_{fin}$. In this case, the RNN is learning the mapping of observation and memory node to the distribution over actions as an FSC strategy network: $\hat{\gamma}_{FSC}\colon \mathsf{ObsSeq}^{\mathscr{M}\times\mathscr{A}}_{fin} \times N \to \mathit{Distr}(\mathit{Act})$

In order to extract the memoryless FSC $\mathscr{A}$ from the FSC strategy network $\hat{\gamma}_{\mathrm{FSC}}$, we collect the predicted distributions across the product set of all possible observations $z \in Z$ and all possible memory nodes $n \in N$. From this prediction, the FSC $\mathscr{A}$ is constructed from the action mapping $\gamma(z, n) = \hat{\gamma}_{\mathrm{FSC}}(z, n)$ and the predefined memory update function δ.

Evaluation. We assume that for POMDP $\mathscr{M} = (M, Z, O)$ and specification φ, we have a finite-memory observation-based strategy $\gamma \in \Gamma^{\mathscr{M}}$ as described above. We use the strategy γ to resolve all nondeterminism in $\mathscr{M}$, resulting in the induced MC $\mathscr{M}^\gamma$, see Def. 1. For this MC, we apply model checking, which in polynomial time reveals whether $\mathscr{M}^\gamma \models \varphi$. For the fixed strategy γ we extracted from the strategy network, this provides hard guarantees about the quality of γ regarding φ. As mentioned before, this strategy is only a prediction obtained from the RNN – so the guarantees necessarily do not directly carry over to the strategy network.

3.3 Improving the Represented Strategy

We describe how we compute a *local improvement* for a strategy that does not satisfy the specification. In particular, we have POMDP $\mathscr{M} = (M, Z, O)$, specification φ, and the strategy $\gamma \in \Gamma^{\mathscr{M}}$ with $\mathscr{M}^{\gamma} \not\models \varphi$. We now create diagnostic information on why the specification is not satisfied.

First, without loss of generality, we assume $\varphi = \mathbb{P}_{\leq\lambda}(\psi)$. Let $\gamma(z)(a)$ denote the probability of choosing action $a \in Act$ upon observation $z \in Z$, under the strategy γ. Let $\Pr^*(s)$ denote the probability to satisfy ψ within the induced MC $\mathscr{M}^{\gamma}$. For some threshold $\lambda' \in [0,1]$, a state $s \in S$ is *critical* iff $\Pr^*(s) > \lambda'$. We define λ' as a function $\lambda' \colon S \times \lambda \to \mathbb{R}$ with respect to the threshold λ from the original specification and the state s. We define the set of critical decision under the strategy γ.

Definition 5 (Critical Decision) *A probability $\gamma(z)(a) > 0$ according to an observation-based strategy $\gamma \in \Gamma$ is a* critical decision *iff there exist states $s, s' \in S$ with $s \in O^{-1}(z)$, $\mathscr{P}(s,a,s') > 0$, and s' is critical.*

Intuitively, a decision is critical if it may lead to a critical state. The set of critical decisions serves as a *counterexample*, generated by the set of critical states and the strategy γ. Note that even if a specification is satisfied for γ, the sets of critical decisions and states may still be non-empty as they depend on the definition of the criticality-threshold λ'.

For each observation $z \in O$ with a critical decision, we construct an optimization problem that minimizes the number of different (critical) actions the strategy chooses per observation class. In particular, the probabilities of action choices under γ are redistributed such that the critical choices are minimized.

$$\max_{\gamma(z)(a), a \in Act} \min_{s \in S} p_s \tag{1}$$

subject to

$$\forall s \in O^{-1}(z). \quad p_s = \sum_{a \in Act} \gamma(z)(a) \cdot \sum_{s' \in S} \mathscr{P}(s,a,s') \cdot p^*(s')$$

Basically, we maximize over the minimal possible worst case probability for critical states, using the original probability p^*. From the resulting improved strategy, we generate a new set of paths starting from the critical states. After converting these new paths into observation-action sequences, we retrain the RNN. By gathering more data from these apparently critical situations, we locally improve the quality of the strategies at those locations and gradually introduce observation-dependencies.

3.4 Correctness and Termination

Correctness of our approach is ensured by evaluating the extracted strategy on the POMDP using model checking. As the investigated problem is undecidable for POMDPs [Madani *et al.*, 1999], our approach is naturally *incomplete*. In order to enforce termination after finite time, we abort the refinement loop after a specified number of iterations, or as soon as the progress from one iteration to the next (in terms of the model checking results) falls below a user-specified threshold.

4 Experimental Results

We evaluate our RNN-based synthesis procedure on benchmark examples that are subject to either LTL specifications or expected cost specifications. For the former, we compare to the tool PRISM-POMDP, and for the latter we compare to PRISM-POMDP and the point-based solver SolvePOMDP [Walraven and Spaan, 2017]. Recall that, in general, a strategy over the continuous belief space induces an infinite memory strategy for POMDPs. PRISM-POMDP employs a discretization (we chose the default level of discretization) of that belief space which technically induces a finite-memory strategy. Therefore solutions from PRISM-POMDP are approximate; the tool computes an upper and lower bound on the optimum.

We selected the two solvers from different research communities because they provide the possibility for a straightforward adaption to our benchmark setting. In particular, the tools support undiscounted rewards and have a simple and similar input interface. Extended experiments with, for instance, Monte-Carlo-based methods [Silver and Veness, 2010] are interesting but beyond the scope of this paper.

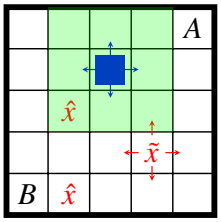
For a fair comparison, instead of terminating our synthesis procedure once a specification is satisfied, we always iterate 10 times, where one iteration encompasses the (re-)training of the RNN, the strategy extraction, the evaluations, and the strategy improvement as detailed in Sect. 3. For instance, for a specification $\varphi = \mathbb{P}_{\leq\lambda}(\psi)$, we leave the λ open and seek to compute $\mathbb{P}_{\min}(\psi)$, that is, we compute the minimal probability of satisfying ψ to obtain a strategy that satisfies φ. We cannot guarantee to reach that optimum, but we rather improve as far as possible within the predefined 10 iterations. The notions are similar for $\mathbb{P}_{\geq\lambda}$ and $\mathbb{P}_{\max}$ as well as for expected cost measures $\mathbb{E}_{\leq\lambda}$ ($\mathbb{E}_{\geq\lambda}$) and $\mathbb{E}_{\min}$ ($\mathbb{E}_{\max}$).

We will now shortly describe our experimental setup and present detailed results for both types of examples.

Implementation and setup. We employ the following Python toolchain to realize the full RNN-based synthesis procedure. First, we use the deep learning library Keras [Chollet and others, 2015] to train the strategy network. To evaluate strategies, we employ the probabilistic model checkers PRISM (LTL) and STORM (undiscounted expected rewards).We evaluated on a 2.3 GHz machine with a 12 GB memory limit and a specified maximum computation time of 10^5 seconds.

4.1 Temporal Logic Examples

We examined three problem settings involving motion planning with LTL specifications. For each of the settings, we use a standard gridworld formulation of an agent with 4 action choices (cardinal directions of movement), see Fig. 2(a). Inside this environment there are a set of static ($\hat{x}$) and moving ($\tilde{x}$) obstacles as well as possible target cells A and B. Each agent has a limited visibility region, indicated by the green area, and can infer its state from observations and knowledge of the environment. We define observations as Boolean functions that take as input the positions of the agent and moving obstacles. Intuitively, the functions describe the 8 possible relative positions of the obstacles with respect to the agent inside its viewing range.



(a)

Problem	$\|S\|$	$\|Act\|$	$\|Z\|$
Navigation (c)	c^4	4	256
Delivery (c)	c^2	4	256
Slippery (c)	c^2	4	256
Maze(c)	$3c+8$	4	7
Grid(c)	c^2	4	2
RockSample[4,4]	257	9	2
RockSample[5,5]	801	10	2
RockSample[7,8]	12545	13	2

(b)

Figure 2: (a) Example environment and (b) Benchmark metrics

1. **Navigation with moving obstacles** – an agent and a single stochastically moving obstacle. The agent task is to maximize the probability to navigate to a goal state A while not colliding with obstacles (both static and moving): $\varphi_1 = \mathbb{P}_{\max}(\neg X \cup A)$ with $x = \hat{x} \cup \tilde{x}$,
2. **Delivery without obstacles** – an agent and static objects (landmarks). The task is to deliver an object from A to B in as few steps as possible: $\varphi_2 = \mathbb{E}_{\min}(\Diamond(A \wedge \Diamond B))$.
3. **Slippery delivery with static obstacles** – an agent where the probability of moving perpendicular to the desired direction is 0.1 in each orientation. The task is to maximize the probability to go back and forth from locations A and B without colliding with the static obstacles $\hat{x}$: $\varphi_3 = \mathbb{P}_{\max}(\Box\Diamond A \wedge \Box\Diamond B \wedge \neg\Diamond X)$, with $x = \hat{x}$,

Evaluation

Fig. 3 compares the size of counterexample in relation to the probability of satisfying an LTL formula in each iteration of the synthesis procedure. In particular, we depict the size of the set $S' \subset S$ of critical states regarding $\varphi_1 = \mathbb{P}_{\max}(\neg X \cup A)$ for the *Navigation* example with grid-size 6. Note that even if the probability to satisfy the LTL specification is nearly one (for the initial state of the POMDP), there may still be critical intermediate states. As can be seen in the figure, while the probability to satisfy the LTL formula increases, the size of the counterexample decreases. In particular, the local improvement (Eq. 1, Sect. 3.3) is demonstrated to be effective.

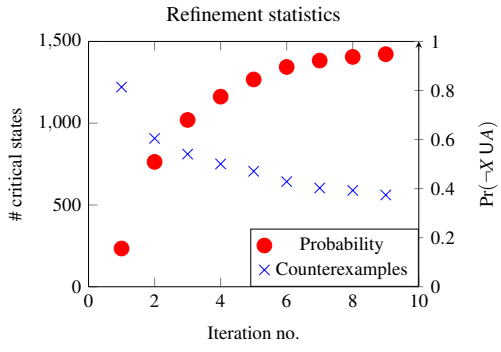


Figure 3: Progression of the number of critical states and the probability of satisfying an LTL specification as a result of local improvement steps.

Problem	States	Type, φ	RNN-based Synthesis Res.	RNN-based Synthesis Time (s)	PRISM-POMDP Res.	PRISM-POMDP Time (s)
Navigation (3)	333	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	0.74	**14.16**	**0.84**	73.88
Navigation (4)	1088	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	0.82	**22.67**	**0.93**†	1034.64
Navigation (4) [2-FSC]	13373	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	0.91	47.26	–	–
Navigation (4) [4-FSC]	26741	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	0.92	59.42	–	–
Navigation (4) [8-FSC]	53477	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	**0.92**	85.26	–	–
Navigation (5)	2725	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	0.91	**34.34**	MO	MO
Navigation (5) [2-FSC]	33357	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	0.92	115.16	–	–
Navigation (5) [4-FSC]	66709	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	0.92	159.61	–	–
Navigation (5) [8-FSC]	133413	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	**0.92**	250.91	–	–
Navigation (10)	49060	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	0.79	**822.87**	MO	MO
Navigation (10) [2-FSC]	475053	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	0.83	1185.41	–	–
Navigation (10) [4-FSC]	950101	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	**0.85**	1488.77	–	–
Navigation (10) [8-FSC]	1900197	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	0.81	1805.22	–	–
Navigation (15)	251965	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	**0.91**	**1271.80***	MO	MO
Navigation (20)	798040	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	**0.96**	**4712.25***	MO	MO
Navigation (30)	4045840	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	**0.95**	**25191.05***	MO	MO
Navigation (40)	–	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_1	TO	TO	MO	MO
Delivery (4) [2-FSC]	80	$\mathbb{E}^{\mathscr{M}}_{\min}$, φ_2	6.02	35.35	**6.0**	**28.53**
Delivery (5) [2-FSC]	125	$\mathbb{E}^{\mathscr{M}}_{\min}$, φ_2	8.11	**78.32**	**8.0**	102.41
Delivery (10) [2-FSC]	500	$\mathbb{E}^{\mathscr{M}}_{\min}$, φ_2	**18.13**	**120.34**	MO	MO
Slippery (4) [2-FSC]	460	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_3	0.78	67.51	**0.90**	**5.10**
Slippery (5) [2-FSC]	730	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_3	0.89	84.32	**0.93**	**83.24**
Slippery (10) [2-FSC]	2980	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_3	**0.98**	**119.14**	MO	MO
Slippery (20) [2-FSC]	11980	$\mathbb{P}^{\mathscr{M}}_{\max}$, φ_3	**0.99**	**1580.42**	MO	MO

Table 1: Synthesizing strategies for examples with LTL specs.

Table 1 contains the results for the above LTL examples. Note that the sizes of the FSCs were included to demonstrate the trade-off between computational tractability and expressivity: a larger FSC means that the strategy can store more information, which may lead to better choices. However, larger FSCs require more computational effort and may require more data for training the RNN. We convey this trade-off in the experiments, as the size of the FSC is often problem-specific. Naturally the strategies produced by the procedure will not have higher maximum probabilities (or lower minimum expected cost) than those generated by the PRISM-POMDP tool. However, they scale for significantly larger environments and settings. In the larger environments (*Navigation*(15) and upwards indicated by a star) we employ the sampling technique outlined at the end of Sect. 3.1 on a dataset with grid-size 10. The strategy still scales to these larger environments even when trained on data from a smaller state space.

Also in Table 1, we compare the effect of increasing the value of k for several k-FSCs. In smaller instances with grid-sizes of 4 and 5, memory-based strategies significantly outperform memoryless ones in terms of quality (the resulting probability or expected cost) while not consuming significantly more time. The increase in performance is due to additional expressiveness of an FSC-based strategy in these environments with a higher density of obstacles.

Summarized, our method scales to significantly larger domains than PRISM-POMDP with competitive computation times. As mentioned before, there is an inherent level of randomness in extracting a strategy. While we always take the first shot result for our experiments, the quality of strategies may improved by sampling several RNN predictions.

†Output was a bound; we give the worst-case value from bound.

Problem	Type	RNN-based Synthesis States	Res	Time (s)	PRISM-POMDP Res	Time (s)	pomdpSolve Res	Time (s)
Maze (1)	$\mathbb{E}^{\mathscr{M}}_{\min}$	68	4.31	31.70	**4.30**	**0.09**	4.30	0.30
Maze (2)	$\mathbb{E}^{\mathscr{M}}_{\min}$	83	5.31	46.65	5.23	2.176	**5.23**	**0.67**
Maze (3)	$\mathbb{E}^{\mathscr{M}}_{\min}$	98	8.10	58.75	7.13	38.82	**7.13**	**2.39**
Maze (4)	$\mathbb{E}^{\mathscr{M}}_{\min}$	113	11.53	58.09	8.58	543.06	**8.58**	**7.15**
Maze (5)	$\mathbb{E}^{\mathscr{M}}_{\min}$	128	14.40	**68.09**	13.00†	4110.50	**12.04**	132.12
Maze (6)	$\mathbb{E}^{\mathscr{M}}_{\min}$	143	22.34	**71.89**	MO	MO	**18.52**	1546.02
Maze (10)	$\mathbb{E}^{\mathscr{M}}_{\min}$	203	100.21	**158.33**	MO	MO	MO	MO
Grid (3)	$\mathbb{E}^{\mathscr{M}}_{\min}$	165	2.90	38.94	2.88	2.332	**2.88**	**0.07**
Grid (4)	$\mathbb{E}^{\mathscr{M}}_{\min}$	381	4.32	79.99	4.13	1032.53	**4.13**	**0.77**
Grid (5)	$\mathbb{E}^{\mathscr{M}}_{\min}$	727	6.62	91.42	MO	MO	**5.42**	**1.94**
Grid (10)	$\mathbb{E}^{\mathscr{M}}_{\min}$	5457	**13.63**	**268.40**	MO	MO	MO	MO
RockSample[4,4]	$\mathbb{E}^{\mathscr{M}}_{\max}$	2432	17.71	35.35	N/A	N/A	**18.04**	**0.43**
RockSample[5,5]	$\mathbb{E}^{\mathscr{M}}_{\max}$	8320	18.40	**43.74**	N/A	N/A	**19.23**	621.28
RockSample[7,8]	$\mathbb{E}^{\mathscr{M}}_{\max}$	166656	20.32	**860.53**	N/A	N/A	**21.64**†	20458.41

Table 2: Comparison for standard POMDP examples.

4.2 Comparison to Existing POMDP Examples

For comparison to existing benchmarks, we extend two examples from PRISM-POMDP for an arbitrary-sized structure: *Maze*(c) with $c+2$ rows and *Grid*(c) – a square grid with length c. We also compare to *RockSample* [Silver and Veness, 2010] (see Table 2(b) for problem metrics).

These problems are quite different to the LTL examples, in particular the significantly smaller observation spaces. As a result, a simple memoryless strategy is insufficient for a useful comparison. For each problem, the size of the k-FSC used is given by: Maze(c) has $k = (c+1)$; Grid(c) has $k = (c-1)$ and *RockSample* with b rocks has $k = b$.

Our method compares favorably with PRISM-POMDP and pomdpSolve for Maze and Grid (Table 2). However, the proposed method performs poorly in comparison to pomdpSolve for *RockSample*: An observation is received after taking an action to *check* a particular rock. This action is never sampled in the modified trajectory-tree based sampling method (Sect. 3.1). Note that our main aim is to enable the efficient synthesis of strategies under linear temporal logic constraints.

5 Summary and Future Work

We introduced a new RNN-based strategy synthesis method for POMDPs and LTL specifications. While we cannot guarantee optimality, our approach shows results that are often close to the actual optimum with competitive computation times for large problem domains.

For the future, we are interested in extending our method to continuous state spaces together with abstraction techniques that would enable to employ our model-based method.

Acknowledgements

This work was partially supported by the grants DARPA D19AP00004 and ONR N00014-18-1-2829.

References

[Amato *et al.*, 2010] Christopher Amato, Daniel S Bernstein, and Shlomo Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *AAMAS*, 21(3):293–320, 2010.

[Baier and Katoen, 2008] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[Chatterjee *et al.*, 2015] Krishnendu Chatterjee, Martin Chmelík, Raghav Gupta, and Ayush Kanodia. Qualitative analysis of POMDPs with temporal logic specifications for robotics applications. In *ICRA*, pages 325–330, 2015.

[Chatterjee *et al.*, 2016] Krishnendu Chatterjee, Martin Chmelík, and Mathieu Tracol. What is decidable about partially observable Markov decision processes with ω-regular objectives. *Journal of Computer and System Sciences*, 82(5):878–911, 2016.

[Chollet and others, 2015] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015. Accessed on 02.13.19.

[Cubuktepe *et al.*, 2017] Murat Cubuktepe, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen, Ivan Papusha, Hasan A. Poonawala, and Ufuk Topcu. Sequential convex programming for the efficient verification of parametric MDPs. In *TACAS (2)*, volume 10206 of *LNCS*, pages 133–150, 2017.

[Cubuktepe *et al.*, 2018] Murat Cubuktepe, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen, and Ufuk Topcu. Synthesis in pmdps: A tale of 1001 parameters. In *ATVA*, volume 11138 of *Lecture Notes in Computer Science*, pages 160–176. Springer, 2018.

[Dehnert *et al.*, 2017] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *CAV (2)*, volume 10427 of *LNCS*, pages 592–600. Springer, 2017.

[Haesaert *et al.*, 2018] Sofie Haesaert, Petter Nilsson, Cristian Ioan Vasile, Rohan Thakker, Ali-akbar Aghamohammadi, Aaron D. Ames, and Richard M. Murray. Temporal logic control of POMDPs via label-based stochastic simulation relations. In *ADHS*, volume 51(16) of *IFAC-PapersOnLine*, pages 271–276. Elsevier, 2018.

[Hahn *et al.*, 2010] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric Markov models. *Software Tools for Technology Transfer*, 13(1):3–19, 2010.

[Hausknecht and Stone, 2015] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable MDPs. *CoRR, abs/1507.06527*, 7(1), 2015.

[Hauskrecht, 2000] Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *J. Artif. Intell. Res.*, 13:33–94, 2000.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[Junges *et al.*, 2018] Sebastian Junges, Nils Jansen, Ralf Wimmer, Tim Quatmann, Leonore Winterer, Joost-Pieter Katoen, and Bernd Becker. Finite-state controllers of POMDPs via parameter synthesis. In *UAI*, 2018.

[Junges *et al.*, 2019] Sebastian Junges, Erika Ábrahám, Christian Hensel, Nils Jansen, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. Parameter synthesis for markov models. *CoRR*, abs/1903.07993, 2019.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1):99–134, 1998.

[Kearns *et al.*, 2000] Michael J Kearns, Yishay Mansour, and Andrew Y Ng. Approximate planning in large POMDPs via reusable trajectories. In *NIPS*, pages 1001–1007, 2000.

[Kingma and Ba, 2014] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint 1412.6980*, 2014.

[Kwiatkowska *et al.*, 2011] Marta 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.

[Littman *et al.*, 2017] Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via GLTL. *arXiv preprint 1704.04341*, 2017.

[Madani *et al.*, 1999] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *AAAI*, pages 541–548. AAAI Press, 1999.

[Meuleau *et al.*, 1999] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *UAI*, pages 427–436. Morgan Kaufmann, 1999.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Norman *et al.*, 2017] Gethin Norman, David Parker, and Xueyi Zou. Verification and control of partially observable probabilistic systems. *Real-Time Systems*, 53(3):354–402, 2017.

[Papadimitriou and Tsitsiklis, 1987] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[Parisotto and Salakhutdinov, 2018] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *ICLR*. OpenReview.net, 2018.

[Pascanu *et al.*, 2013] Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *CoRR*, abs/1312.6026, 2013.

[Pineau *et al.*, 2003] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, pages 1025–1032. Morgan Kaufmann, 2003.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

[Pritzel *et al.*, 2017] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *ICML*, volume 70 of *Proc. of Machine Learning Research*, pages 2827–2836. PMLR, 2017.

[Santoro *et al.*, 2018] Adam Santoro, Ryan Faulkner, David Raposo, Jack W. Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy P. Lillicrap. Relational recurrent neural networks. In *NeurIPS*, pages 7310–7321, 2018.

[Silver and Veness, 2010] David Silver and Joel Veness. Monte-carlo planning in large POMDPs. In *NIPS*, pages 2164–2172, 2010.

[Sutton *et al.*, 2000] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063, 2000.

[Vlassis *et al.*, 2012] Nikos Vlassis, Michael L. Littman, and David Barber. On the computational complexity of stochastic controller optimization in POMDPs. *ACM Trans. on Computation Theory*, 4(4):12:1–12:8, 2012.

[Walraven and Spaan, 2017] Erwin Walraven and Matthijs Spaan. Accelerated vector pruning for optimal POMDP solvers. In *AAAI*, pages 3672–3678. AAAI Press, 2017.

[Wang *et al.*, 2018] Yue Wang, Swarat Chaudhuri, and Lydia E. Kavraki. Bounded policy synthesis for pomdps with safe-reachability objectives. In *AAMAS*, pages 238–246. Int'l Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018.

[Wierstra *et al.*, 2007] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *ICANN*, pages 697–706. Springer, 2007.

[Wimmer *et al.*, 2014] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Minimal counterexamples for linear-time probabilistic verification. *Theoretical Computer Science*, 549:61–100, 2014.