Counterexample-Guided Strategy Improvement for POMDPs
Using Recurrent Neural Networks

Steven Carr\textsuperscript{1}, Nils Jansen\textsuperscript{2}, Ralf Wimmer\textsuperscript{3,4},
Alexandru C. Serban,\textsuperscript{2} Bernd Becker\textsuperscript{3} and Ufuk Topcu\textsuperscript{1}

\textsuperscript{1}The University of Texas at Austin
\textsuperscript{2}Radboud University, Nijmegen, The Netherlands
\textsuperscript{3}Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany
\textsuperscript{4}Concept Engineering GmbH, Freiburg im Breisgau, Germany

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 partial 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 surpass 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, [Hauskrecht 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 \( \mathbb{R} \) is denoted by \( \text{Distr} \).

The set of all distributions on \( \mathbb{R} \) is denoted by \( \text{Distr} \).

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 \( \text{Distr}(X) \). The support of a distribution \( \mu \) is \( \text{supp}(\mu) = \{ x \in X | \mu(x) > 0 \} \).

\( \text{POMDPs} \). A Markov decision process (MDP) \( M \) is a tuple \( M = (S, \mathcal{A}, \mathcal{O}) \) with a finite (or countably infinite) set \( S \) of states, a finite set \( \mathcal{A} \) of actions, and a transition function \( \mathcal{O} : S \times \mathcal{A} \to \text{Distr}(S) \). We use a reward function \( r : S \times \mathcal{A} \to \mathbb{R} \). A finite path \( \pi \) of an MDP \( M \) is a sequence of states and actions; last(\( \pi \)) is the last state of \( \pi \). The set of finite paths of \( M \) is \( \text{Paths}^M_{\text{fin}} \). A discrete-time Markov chain (MC) is an MDP with \( |\mathcal{A}(s)| = 1 \) for all \( s \in S \).

A strategy \( \gamma \) for and MDP \( M \) is a function \( \gamma : \text{Paths}^M_{\text{fin}} \to \text{Distr}(\mathcal{A}) \) with \( \text{supp}(\gamma(\pi)) \subseteq \text{Act}(\text{last}(\pi)) \) for all \( \pi \in \text{Paths}^M_{\text{fin}} \). A strategy \( \gamma \) is memoryless if last(\( \pi \)) is last(\( \pi' \)) implies \( \gamma(\pi) = \gamma(\pi') \) for all \( \pi, \pi' \in \text{Paths}^M_{\text{fin}} \).

Definition 1 (Induced Markov Chain) For a MDP \( M = (S, \mathcal{A}, \mathcal{O}) \) and a strategy \( \gamma \in \Gamma^M \), the MC induced by \( M \) and \( \gamma \) is given by \( M^\gamma = (\text{Paths}^M_{\text{fin}}, P^\gamma) \) where:

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

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

The set of all finite observation-action sequences for a POMDP \( \mathcal{M} \) is denoted by \( \text{ObsSeq}^M_{\text{fin}} \).

* Also referred to as stochastic strategy or policy.
Definition 3 (POMDP Strategy) An observation-based strategy for a POMDP $\mathcal{M}$ is a function $\gamma: \text{ObsSeq}_{\text{fin}} \rightarrow \text{Distr}(\text{Act})$ such that $\text{supp}(\gamma(O(\pi))) \subseteq \text{Act} \left(\text{last}(\pi)\right)$, for all $\pi \in \text{Paths}_{\text{fin}}^{\mathcal{M}}$. $\Gamma_{\#}^\mathcal{M}$ is the set of observation-based strategies for $\mathcal{M}$.

A memoryless observation-based strategy $\gamma \in \Gamma_{\#}^\mathcal{M}$ is analogous to a memoryless MDP strategy, formally we simplify to $\gamma: Z \rightarrow \text{Distr}(\text{Act})$, 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 (FSCs) [Meuleau et al., 1999].

Definition 4 (FSC) A $k$-FSC for a POMDP is a tuple $\mathcal{A} = (N, n_\text{init}, \gamma, \delta)$ where $N$ is a finite set of $k$ memory nodes, $n_\text{init} \in N$ is the initial memory node, $\gamma$ is the action mapping $\gamma: N \times Z \rightarrow \text{Distr}(\text{Act})$ and $\delta$ is the memory update $\delta: N \times Z \times \text{Act} \rightarrow N$.

Let $\gamma_{\#} \in \Gamma_{\#}^\mathcal{M}$ 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 $\gamma$ is left out of the product. A memoryless strategy $\gamma \in \Gamma_{\#}^\mathcal{M} \times \mathcal{A}$ then determines the action mapping and can be projected to the finite-memory strategy $\gamma_{\#} \in \Gamma_{\#}^\mathcal{M}$.

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:

$$\Psi ::= a \mid (\Psi \land \Psi) \mid \neg \Psi \mid \bigcirc \Psi \mid \bigboxdot \Psi \mid (\Psi \cup \Psi).$$

Intuitively, a path $\pi$ satisfies the proposition $a$ if its first state does; $(\Psi_1 \land \Psi_2)$ is satisfied, if $\pi$ satisfies both $\Psi_1$ and $\Psi_2$; $\neg \Psi$ is true on $\pi$ if $\Psi$ is not satisfied. The formula $\bigcirc \Psi$ holds on $\pi$ if the subpath starting at the second state of $\pi$ satisfies $\Psi$. The path $\pi$ satisfies $\bigboxdot \Psi$ if all suffixes of $\pi$ satisfy $\Psi$. Finally, $\pi$ satisfies $(\Psi_1 \cup \Psi_2)$ if there is a suffix of $\pi$ that satisfies $\Psi_2$ and all longer suffixes satisfy $\Psi_1$. $\bigcirc \Psi$ abbreviates (true $\Psi$).

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

If $\phi$ (either LTL or expected reward specification) is satisfied in a (PO)MDP $\mathcal{M}$ under $\gamma$, we write $\mathcal{M}^\gamma \models \phi$, 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 $\gamma$, checking whether $\mathcal{M}^\gamma \models \phi$ 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

![Figure 1: Flowchart of the RNN-based refinement loop](image-url)

**Formal Problem Statement.** For a POMDP $\mathcal{M}$ and a specification $\phi$, where either $\phi = F_{\mathcal{A}}(\Psi)$ with $\Psi$ an LTL formula, or $\phi = E_{\mathcal{A}}(\Psi)$, the problem is to determine a (finite-memory) strategy $\gamma \in \Gamma_{\#}^\mathcal{M}$ such that $\mathcal{M}^\gamma \models \phi$.

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 $\phi$ 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 a linear programming (LP) approach that locally improves action choices of the current strategy at these critical states, see
We iterate this procedure until the specification is satisfied
for different problem sizes; such a framework can provide 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 memory.

**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 $\hat{\gamma}: \text{ObsSeq}_{\text{fin}} \rightarrow \text{Distr}(\text{Act})$). 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 ObsSeq$_{\text{fin}}$, the model learns a strategy $\hat{\gamma} \in \Gamma_{\text{fin}}$. The output is a discrete probability distribution over the actions $\text{Act}$, 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 $\mathcal{M} = (M, Z, O)$ and a specification $\phi$, instead of randomly generating observation sequences, we first compute a strategy $\gamma \in \Gamma^M$ of the underlying MDP $M$ that satisfies $\phi$. Then we sample uniformly over all states of the MDP and generate finite paths (of a fixed maximal length) from Path$_{\text{fin}}^M$ of the induced MC $M'$, thereby creating multiple trajectory trees. For each finite path $\pi \in \text{Path}_{\text{fin}}^M$, we generate one possible observation-action sequence $\pi(z)$ in ObsSeq$_{\text{fin}}^\phi$ such that $\pi(z) = z_0, a_0, \ldots, z_n$, with $z_i = O(\pi[i])$, where $\pi[i]$ denotes the $i$-th state of $\pi$ for all $1 \leq i \leq n$. We form the training set $\mathcal{D}$ 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 $\mathcal{M}$ with a large state space ($|\mathcal{S}| > 10^5$), 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 $\text{Act}$ with $\mathcal{M}$. 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 $\mathcal{M}$, we use the trained strategy network $\gamma: \text{ObsSeq}_{\text{fin}} \rightarrow \text{Distr}(\text{Act})$ 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 $\phi = (N, n_1, \gamma, \delta)$ as in Def. 4. We first redefine the memory update function $\delta$ in a problem-specific way, for instance, $\delta$ changes the memory node when an observation is repeated. Consider observation sequence $\pi \in \text{ObsSeq}_{\text{fin}}^\phi$ 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 $\pi$. We define $\delta(n_k, z_i, a_i) = n_{k+1}$, if $\pi[i] = (z_i, a_i)$, and there exists a $j < i$ such that $\pi[j] = (z_j, a_j)$ with $z_i = z_j$. Similarly, we account for specific memory choices akin to the relevant LTL specification.

Once $\delta$ has been defined, we compute a product POMDP $\mathcal{M} \times \phi$ 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 ObsSeq$_{\text{fin}}^\phi$, we generate observation-node-action sequences $(z_0, n_0), (a_0, \ldots, a_{n-1}, (z_n, n_n))$ from ObsSeq$_{\text{fin}}^\phi \times \phi$. In this case, the RNN is learning the mapping of observation and memory node to the distribution over actions as an FSC strategy network $\hat{\gamma}_{\text{FSC}}: \text{ObsSeq}_{\text{fin}}^\phi \times \phi \times N \rightarrow \text{Distr}(\text{Act})$

In order to extract the memoryless FSC $\phi$ from the FSC strategy network $\hat{\gamma}_{\text{FSC}}$, 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 $\phi$ is constructed from the action mapping $\gamma(z, n) = \hat{\gamma}_{\text{FSC}}(z, n)$ and the predefined memory update function $\delta$.

**Evaluation.** We assume that for POMDP $\mathcal{M} = (M, Z, O)$ and specification $\phi$, we have a finite-memory observation-based strategy $\gamma \in \Gamma_{\text{fin}}^M$ as described above. We use the strategy $\gamma$ to resolve all nondeterminism in $\mathcal{M}$, resulting in the induced MC $\mathcal{M}^\top$, see Def. 1. For this MC, we apply model checking, which in polynomial time reveals whether $\mathcal{M}^\top \models \phi$. For the fixed strategy $\gamma$ we extracted from the strategy network, this provides hard guarantees about the quality of $\gamma$ regarding $\phi$. As mentioned before, this 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 $\mathcal{M} = (M, Z, O)$, specification $\phi$, and the strategy $\gamma \in \Gamma^M$ with $\mathcal{M}^\top \not\models \phi$. We now create diagnostic information on why the specification is not satisfied.
We define the set of critical decision under the strategy $\gamma$ of our approach is ensured by evaluating the correctness and Spaan, 2017]. Recall that, in general, a strategy over the POMDP and the point-based solver SolvePOMDP [Walraven and Spaan, 2017] 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 ($\tilde{x}$) and moving ($\hat{x}$) 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.

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): $\phi_1 = P_{\text{max}}(\neg X U A)$ with $x = \tilde{x} \cup \hat{x}$,

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: $\phi_2 = P_{\text{min}}(\Diamond(A \land \Diamond B))$.
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 $x$: $\phi = P_{\text{min}}(\square \diamond AP \land \diamond \square BP \land \neg \diamond X)$, with $x = \lambda$.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size</th>
<th>Acf</th>
<th>FSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation (c)</td>
<td>$c^2$</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>Delivery (c)</td>
<td>$c^2$</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>Slippery (c)</td>
<td>$c^2$</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>Maze(c)</td>
<td>$3c + 8$</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Grid(c)</td>
<td>$c^2$</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>RockSample[4,4]</td>
<td>257</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>RockSample[5,5]</td>
<td>801</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>RockSample[7,8]</td>
<td>12545</td>
<td>13</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1: Synthesizing strategies for examples with LTL specs.

<table>
<thead>
<tr>
<th>Problem</th>
<th>States</th>
<th>Type</th>
<th>$\phi$</th>
<th>RNN-based Synthesis</th>
<th>PRISM-POMDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation (3)</td>
<td></td>
<td></td>
<td></td>
<td>0.74</td>
<td>14.16</td>
</tr>
<tr>
<td>Navigation (4)</td>
<td></td>
<td></td>
<td></td>
<td>0.82</td>
<td>22.67</td>
</tr>
<tr>
<td>Navigation (4) [2-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.91</td>
<td>47.26</td>
</tr>
<tr>
<td>Navigation (4) [4-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.92</td>
<td>59.42</td>
</tr>
<tr>
<td>Navigation (4) [8-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.92</td>
<td>85.26</td>
</tr>
<tr>
<td>Navigation (5)</td>
<td></td>
<td></td>
<td></td>
<td>0.91</td>
<td>34.34</td>
</tr>
<tr>
<td>Navigation (5) [2-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.92</td>
<td>115.16</td>
</tr>
<tr>
<td>Navigation (5) [4-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.92</td>
<td>159.61</td>
</tr>
<tr>
<td>Navigation (5) [8-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.92</td>
<td>250.91</td>
</tr>
<tr>
<td>Navigation (10)</td>
<td></td>
<td></td>
<td></td>
<td>0.91</td>
<td>1271.80*</td>
</tr>
<tr>
<td>Navigation (10) [2-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.79</td>
<td>1185.41</td>
</tr>
<tr>
<td>Navigation (10) [4-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.83</td>
<td>1185.41</td>
</tr>
<tr>
<td>Navigation (10) [8-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.85</td>
<td>1488.77</td>
</tr>
<tr>
<td>Navigation (10) [15-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.81</td>
<td>1583.22*</td>
</tr>
<tr>
<td>Navigation (15)</td>
<td></td>
<td></td>
<td></td>
<td>0.91</td>
<td>1271.80*</td>
</tr>
<tr>
<td>Navigation (20)</td>
<td></td>
<td></td>
<td></td>
<td>0.96</td>
<td>4712.25*</td>
</tr>
<tr>
<td>Navigation (30)</td>
<td></td>
<td></td>
<td></td>
<td>0.95</td>
<td>25191.05*</td>
</tr>
<tr>
<td>Navigation (40)</td>
<td></td>
<td></td>
<td></td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>Delivery (4) [2-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.02</td>
<td>38.35</td>
</tr>
<tr>
<td>Delivery (5) [2-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.11</td>
<td>78.32</td>
</tr>
<tr>
<td>Delivery (10) [2-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.13</td>
<td>120.34*</td>
</tr>
<tr>
<td>Slippery (4) [2-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.76</td>
<td>57.51</td>
</tr>
<tr>
<td>Slippery (5) [2-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.89</td>
<td>84.32</td>
</tr>
<tr>
<td>Slippery (10) [2-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.98</td>
<td>119.14*</td>
</tr>
<tr>
<td>Slippery (20) [2-FSC]</td>
<td></td>
<td></td>
<td></td>
<td>0.99</td>
<td>1580.42*</td>
</tr>
</tbody>
</table>

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

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

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 $S' \subseteq S$ of critical states regarding $\phi = P_{\text{max}}(\neg X U A)$ 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.

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.

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

$\hat{\psi} = \psi_{\text{max}}(\square \diamond A \land \diamond \square B \land \neg \diamond X)$, with $x = \lambda$. **Output was a bound; we give the worst-case value from bound.**
We introduced a new RNN-based strategy synthesis method (Sect. 3.1). Note that our main aim is to enable the efficient synthesis of strategies under linear temporal logic constraints. While we cannot guarantee optimality, our approach shows results that are often close for POMDPs and LTL specifications. As shown in Table 2, our method performs favorably 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.

### References


<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>RNN-based Synthesis</th>
<th>PRISM-POMDP</th>
<th>pomdpSolve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maze (1)</td>
<td>E</td>
<td>68</td>
<td>6.31</td>
<td>31.70</td>
</tr>
<tr>
<td>Maze (2)</td>
<td>E</td>
<td>83</td>
<td>5.31</td>
<td>46.65</td>
</tr>
<tr>
<td>Maze (3)</td>
<td>E</td>
<td>98</td>
<td>8.10</td>
<td>58.75</td>
</tr>
<tr>
<td>Maze (4)</td>
<td>E</td>
<td>113</td>
<td>11.53</td>
<td>58.09</td>
</tr>
<tr>
<td>Maze (5)</td>
<td>E</td>
<td>128</td>
<td>14.40</td>
<td>68.09</td>
</tr>
<tr>
<td>Maze (6)</td>
<td>E</td>
<td>143</td>
<td>22.34</td>
<td>71.89</td>
</tr>
<tr>
<td>Maze (10)</td>
<td>E</td>
<td>203</td>
<td>100.21</td>
<td>158.33</td>
</tr>
<tr>
<td>Grid (1)</td>
<td>E</td>
<td>381</td>
<td>4.32</td>
<td>79.99</td>
</tr>
<tr>
<td>Grid (2)</td>
<td>E</td>
<td>727</td>
<td>6.62</td>
<td>91.42</td>
</tr>
<tr>
<td>Grid (10)</td>
<td>E</td>
<td>5457</td>
<td>13.63</td>
<td>268.40</td>
</tr>
<tr>
<td>RockSample</td>
<td>E</td>
<td>8320</td>
<td>18.40</td>
<td>43.74</td>
</tr>
<tr>
<td>RockSample</td>
<td>E</td>
<td>166656</td>
<td>20.32</td>
<td>860.53</td>
</tr>
</tbody>
</table>

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.


