Counterexample-Guided Strategy Improvement for POMDPs Using Recurrent Neural Networks

Machine Learning and Formal Verification Join Forces?

Nils Jansen
IJCAI, August 15, 2019
joint work with:

Steve Carr, Alexandru Serban, Ralf Wimmer, Ufuk Topcu, Bernd Becker
Help the Robot

Find the best way to the rock
Help the Robot

Find the best way to the rock
Help the Robot

Find the best way to the rock

Visit the parachute on the way
Help the Robot

Find the best way to the rock

Visit the parachute on the way

Take expensive surfaces into account
Help the Robot

Find the best way to the rock

Visit the parachute on the way

Take expensive surfaces into account

Avoid randomly moving dust storm
Help the Robot

Find the best way to the **rock**

Visit the **parachute** on the way

Take **expensive surfaces** into account

Avoid randomly moving **dust storm**

Find safe and/or cost-optimal strategy to get to the **airbag**
Help the Robot

Find the best way to the rock

Visit the parachute on the way

Take expensive surfaces into account

Avoid randomly moving dust storm

Find safe and/or cost-optimal strategy to get to the airbag

Underlying Model: Markov Decision Process

$Pr_{max}(\neg s \ U (\Diamond(p \land \Box \Diamond r)))$

$EC_{min}(\Diamond r)$

temporal logic

expected cost

Nils Jansen

Radboud University
Partial Observability

It’s a well known fact that you must spin a USB three times before it will fit. From this, we can gather that a USB has three states:

- Up position
- Down position
- Superposition

Until the USB is observed it will stay in the superposition. Therefore it will not fit until observed - except for in cases of USB tunnelling.
Help the Robot with Partial Observability

Robot has restricted range of vision

Find safe and/or cost-optimal strategy to get to the airbag
Help the Robot with Partial Observability

Robot has restricted range of vision

Storm is only observable when near

Find safe and/or cost-optimal strategy to get to the airbag
Help the Robot with Partial Observability

Robot has restricted range of vision

Storm is only **observable** when near

For robot, storm is either **near** or **far**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Find safe and/or cost-optimal strategy to get to the airbag
Help the Robot with Partial Observability

Robot has restricted range of vision

Storm is only *observable* when near

For robot, storm is either **near** or **far**

Belief state: Likelihood of the actual position of the storm

Find safe and/or cost-optimal strategy to get to the airbag

infinite belief MDP

Nils Jansen

Radboud University
POMDPs - Applications

Stock Market

Surveying Threatened Species

Health Care

Wireless Sensor Networks

Autonomous Systems

Machine Vision
Computing Strategies for POMDPs

- Randomized with infinite memory: undecidable, optimal results.
Computing Strategies for POMDPs

- **Randomized with infinite memory**: undecidable, optimal results.
- **Randomized with finite memory**: NP-hard, SQRT-SUM-hard, in PSPACE, not optimal in general, but sufficient for many applications.
Computing Strategies for POMDPs

- **Randomized with infinite memory**: undecidable, optimal results.

- **Randomized with finite memory**: NP-hard, SQRT-SUM-hard, in PSPACE, not optimal in general, but sufficient for many applications.

- Intuitively: Randomization can often trade off memory.

\[ \sigma: \text{ObsSeq}_{\text{fin}} \rightarrow \text{Distr}(\text{Act}) \quad \sigma: \text{Obs} \rightarrow \text{Distr}(\text{Act}) \]
POMDP Solving - State of The Art

Point-based/Approximate

Monte Carlo
POMDP Solving - State of The Art

- Point-based/Approximate
  - Perseus
  - PomdpSolve
  - Sarsop
  - PRISM-POMDP
- Monte Carlo
  - POMCP
POMDP Solving - State of The Art

- Point-based/Approximate:
  - Perseus
  - PomdpSolve
  - Sarsop

- Monte Carlo:
  - PRISM-POMDP
  - POMCP

- Infinite Horizon
- Indefinite Horizon
- Temporal Logic
POMDP Solving - State of The Art

Point-based/Approximate
- Perseus
- PomdpSolve
- Sarsop
- PRISM-POMDP

Monte Carlo
- POMCP

Infinite Horizon
- Indefinite Horizon
- Temporal Logic
POMDP Solving - State of The Art

Point-based/Approximate

PomdpSolve

PRISM-POMDP

Indefinite Horizon

Temporal Logic
The Problem: Strategy Synthesis

$\mathcal{M}$

POMDP

$\varphi$

Specification

Probabilistic Temporal Logic Constraints
The Problem: Strategy Synthesis

POMDP \( \mathcal{M} \) → Specification \( \varphi \) → Probabilistic Temporal Logic Constraints

\( \mathcal{M} \) \( \varphi \) \( \sigma \)

Find Strategy
The Problem: Strategy Synthesis

POMDP → Specification

\( M \)  \( \varphi \)

Probabilistic Temporal Logic Constraints

Find Strategy

\( \sigma \)

Apply Strategy to POMDP

\( M^\sigma \)

Induced Model Satisfies Specification

\( M^\sigma \models \varphi \)
The Problem: Strategy Synthesis

- POMDP
- Specification

Probabilistic Temporal Logic
Constraints

Find Strategy

Apply Strategy to POMDP

Induced Model Satisfies Specification

$\mathcal{M} \models \varphi$
Be Lazy: Guess a Strategy and Verify!

POMDP \xrightarrow{\mathcal{M}} \text{Specification} \xrightarrow{\varphi} \text{Probabilistic Temporal Logic Constraints}

\sigma \xrightarrow{\text{Guess Candidate Strategy}}
Be Lazy: Guess a Strategy and Verify!

- POMDP
- Specification
  - Probabilistic Temporal Logic Constraints
- Guess Candidate Strategy
- Apply Strategy to POMDP

\( \mathcal{M} \quad \phi \quad \varphi \quad \sigma \quad \mathcal{M}^\sigma \)
Be Lazy: Guess a Strategy and Verify!

- POMDP
- Specification
  - Probabilistic Temporal Logic Constraints
  - $\mathcal{M}$
  - $\varphi$
- Guess Candidate Strategy
  - $\sigma$
- Apply Strategy to POMDP
  - $\mathcal{M}^\sigma$
- Model Checking
  - $\mathcal{M}^\sigma \models \varphi$?
  - $\text{SAT}$
- how to guess a good strategy?
  - efficient

Nils Jansen

Radboud University
Let Machine Learning do the Guessing?

- **POMDP**
- **Specification**

how to employ a neural network?

```
M
φ

σ
strategy network

Apply Strategy to POMDP

M^σ
M^σ ⊨ φ?

MODEL CHECKING

UNSAT

SAT
```
RNN Strategy Improvement

M \quad \phi

strategy network
RNN Strategy Improvement

\[ M \quad \varphi \quad \sigma \]

- **POMDP**
- **Specification**
- **Strategy Network**
- **Extract Strategy**
- **Apply Strategy to POMDP**
RNN Strategy Improvement

POMDP

Specification

ℳ

φ

strategy network

σ

ℳσ

Extract Strategy

ℳσ ⊧ φ ?

Apply Strategy to POMDP

SAT

Model Checking

ℳσ ⊧ φ ?
RNN Strategy Improvement

POMDP $\mathcal{M}$

Specification $\varphi$

strategy network

$\sigma$

Extract Strategy

Apply Strategy to POMDP

$\mathcal{M}^\sigma$

$\mathcal{M}^\sigma \models \varphi$?

Model Checking

SAT

$S' \subseteq S$

UNSAT

Counterexamples

Apply Strategy to POMDP

Nils Jansen
RNN Strategy Improvement

- **POMDP**
- **Specification**

**ℳ**

**φ**

strategy network

- **Extract Strategy**
- **Apply Strategy to POMDP**

- **ℳ^σ**

- **ℳ^σ ⊨ φ?**

- **SAT**

- **UNSAT**

- **S' ⊆ S**

- **Counterexamples**

- **Local Improvement of the Strategy**
RNN Strategy Improvement

POMDP \( \mathcal{M} \) → Specification \( \varphi \)

strategy network

Extract Strategy

Apply Strategy to POMDP

\( \mathcal{M}^\sigma \) → \( \mathcal{M}^\sigma \models \varphi ? \)

Model Checking

Training Data

Observation-Action Sequences

Local Improvement of the Strategy

\( K \)

\( S' \subseteq S \)

Counterexamples

\( SAT \)

\( UNSAT \)
Learning Strategies with RNNs
Learning Strategies with RNNs

Recurrent Neural Network
- long short-term memory (LSTM) architecture to learn dependencies in sequential data
- trained with observation-action sequences $ObsSeq_{fin}$
- strategy network $\sigma: ObsSeq_{fin} \rightarrow Distr(Act)$
- observations are input labels, actions are output labels
Learning Strategies with RNNs

Recurrent Neural Network
- long short-term memory (LSTM) architecture to learn dependencies in sequential data
- trained with observation-action sequences $\text{ObsSeq}_{\text{fin}}$
- strategy network $\sigma : \text{ObsSeq}_{\text{fin}} \rightarrow \text{Distr(Act)}$
- observations are input labels, actions are output labels

Initial Training
- compute optimal MDP strategy
- generate (possible) observation-action sequences
Improving the Strategy
Improving the Strategy

• Identify **critical decisions** that lead to states with high probability of **violating** the specification.
Improving the Strategy

- Identify **critical decisions** that lead to states with high probability of **violating** the specification.
- For each observation with critical decision, **minimize** the number of different critical actions.

Local linear program

\[
\begin{align*}
\max_{\gamma(z) \in \mathcal{A}} \min_{a \in \mathcal{A}, s \in S} p_s \\
\text{subject to} \\
\forall s \in O^{-1}(z). \quad p_s = \sum_{a \in \mathcal{A}} \gamma(z) \cdot \sum_{s' \in S} \mathcal{P}(s, a, s') \cdot p^*(s')
\end{align*}
\]
Improving the Strategy

- Identify **critical decisions** that lead to states with high probability of **violating** the specification.
- For each observation with critical decision, **minimize** the number of different critical actions.
- **Retrain** with the new (locally improved) strategy.

Local linear program

\[
\begin{align*}
\max_{\gamma(z)(a), a \in \text{Act}} & \min_{s \in S} p_s \\
\text{subject to} & \\
\forall s \in O^{-1}(z). & p_s = \sum_{a \in \text{Act}} \gamma(z)(a) \cdot \sum_{s' \in S} \mathcal{P}(s, a, s') \cdot p^*(s')
\end{align*}
\]
Improving the Strategy

• Identify **critical decisions** that lead to states with high probability of **violating** the specification.

• For each observation with critical decision, **minimize** the number of different critical actions.

• **Retrain** with the new (locally improved) strategy.

\[
\max_{\gamma(z)(a), a \in \text{Act}} \min_{s \in \mathcal{S}} p_s \\
\text{subject to} \\
\forall s \in O^{-1}(z). \quad p_s = \sum_{a \in \text{Act}} \gamma(z)(a) \cdot \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') \cdot p^*(s')
\]
Finite-memory Strategies (FSC)

- Encode **finite memory** directly into the state space:

- Strategy network is of the form \( \sigma: \text{ObsSeq}_{\text{fin}} \rightarrow \text{Distr(Act)} \)

- But: How to infer a **memory-update function** to construct an FSC?
Finite-memory Strategies (FSC)

- Encode finite memory directly into the state space:

  ![Diagram of state transitions]

- Strategy network is of the form $\sigma : ObsSeq_{fin} \rightarrow Distr(Act)$

- But: How to infer a memory-update function to construct an FSC?

- Predefine memory update, for instance (deterministic) transition upon repetition of an observation.

- Compute product of FSC and POMDP and compute memoryless strategy.
Correctness and Completeness?

- POMDP
- Specification
- Apply Strategy to POMDP
- Model Checking
- Training Data
- Local proof
- Counterexamples

$\mathcal{M}$

$\phi$

$\mathcal{M}^\sigma$ $\models \phi$?

$\sigma$

$S' \subseteq S$
Correctness and Completeness?

Correct, as each strategy prediction is evaluated using model checking.
Correctness and Completeness?

Correct, as each strategy prediction is evaluated using model checking.

Not complete, as we may never find a feasible strategy. Also, problem is undecidable (or hard) anyways :).
## Experiments - LTL

| Problem          | $|S|$ | $|Act|$ | $|Z|$ |
|------------------|-----|------|-----|
| Navigation (c)   | $c^4$ | 4  | 256 |
| Delivery (c)     | $c^2$ | 4  | 256 |
| Slippy (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  |

<table>
<thead>
<tr>
<th>Problem</th>
<th>States</th>
<th>Type, $\varphi$</th>
<th>RNN-based Synthesis</th>
<th>PRISM-POMDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation (3)</td>
<td>333</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.74</td>
<td>14.16</td>
</tr>
<tr>
<td>Navigation (4)</td>
<td>1088</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.82</td>
<td>22.67</td>
</tr>
<tr>
<td>Navigation (4) [2-FSC]</td>
<td>13373</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.91</td>
<td>47.26</td>
</tr>
<tr>
<td>Navigation (4) [4-FSC]</td>
<td>26741</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.92</td>
<td>59.42</td>
</tr>
<tr>
<td>Navigation (4) [8-FSC]</td>
<td>53477</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.92</td>
<td>85.26</td>
</tr>
<tr>
<td>Navigation (5)</td>
<td>2725</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.91</td>
<td>34.34</td>
</tr>
<tr>
<td>Navigation (5) [2-FSC]</td>
<td>33357</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.92</td>
<td>115.16</td>
</tr>
<tr>
<td>Navigation (5) [4-FSC]</td>
<td>66709</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.92</td>
<td>159.61</td>
</tr>
<tr>
<td>Navigation (5) [8-FSC]</td>
<td>133413</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.92</td>
<td>250.91</td>
</tr>
<tr>
<td>Navigation (10)</td>
<td>49060</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.79</td>
<td>822.87</td>
</tr>
<tr>
<td>Navigation (10) [2-FSC]</td>
<td>475053</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.83</td>
<td>1185.41</td>
</tr>
<tr>
<td>Navigation (10) [4-FSC]</td>
<td>950101</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.85</td>
<td>1488.77</td>
</tr>
<tr>
<td>Navigation (10) [8-FSC]</td>
<td>1900197</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.81</td>
<td>1805.22</td>
</tr>
<tr>
<td>Navigation (15)</td>
<td>251965</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.91</td>
<td>1271.80*</td>
</tr>
<tr>
<td>Navigation (20)</td>
<td>798040</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.96</td>
<td>4712.25*</td>
</tr>
<tr>
<td>Navigation (30)</td>
<td>4045840</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>0.95</td>
<td>2519.10*</td>
</tr>
<tr>
<td>Navigation (40)</td>
<td>-</td>
<td>$P_{\text{max}, \varphi_1}$</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>Delivery (4) [2-FSC]</td>
<td>80</td>
<td>$P_{\text{max}, \varphi_2}$</td>
<td>6.02</td>
<td>35.35</td>
</tr>
<tr>
<td>Delivery (5) [2-FSC]</td>
<td>125</td>
<td>$P_{\text{max}, \varphi_2}$</td>
<td>8.11</td>
<td>78.32</td>
</tr>
<tr>
<td>Delivery (10) [2-FSC]</td>
<td>500</td>
<td>$P_{\text{max}, \varphi_2}$</td>
<td>18.13</td>
<td>120.34</td>
</tr>
<tr>
<td>Slippy (4) [2-FSC]</td>
<td>460</td>
<td>$P_{\text{max}, \varphi_3}$</td>
<td>0.78</td>
<td>67.51</td>
</tr>
<tr>
<td>Slippy (5) [2-FSC]</td>
<td>730</td>
<td>$P_{\text{max}, \varphi_3}$</td>
<td>0.89</td>
<td>84.32</td>
</tr>
<tr>
<td>Slippy (10) [2-FSC]</td>
<td>2980</td>
<td>$P_{\text{max}, \varphi_3}$</td>
<td>0.98</td>
<td>119.14</td>
</tr>
<tr>
<td>Slippy (20) [2-FSC]</td>
<td>11980</td>
<td>$P_{\text{max}, \varphi_3}$</td>
<td>0.99</td>
<td>1580.42</td>
</tr>
</tbody>
</table>
# Experiments - Standard POMDPs

<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></td>
<td></td>
<td>States</td>
<td>Res</td>
<td>Time (s)</td>
</tr>
<tr>
<td>Maze (1)</td>
<td>$E_{\min}^M$</td>
<td>68</td>
<td>4.31</td>
<td>31.70</td>
</tr>
<tr>
<td>Maze (2)</td>
<td>$E_{\min}^M$</td>
<td>83</td>
<td>5.31</td>
<td>46.65</td>
</tr>
<tr>
<td>Maze (3)</td>
<td>$E_{\min}^M$</td>
<td>98</td>
<td>8.10</td>
<td>58.75</td>
</tr>
<tr>
<td>Maze (4)</td>
<td>$E_{\min}^M$</td>
<td>113</td>
<td>11.53</td>
<td>58.09</td>
</tr>
<tr>
<td>Maze (5)</td>
<td>$E_{\min}^M$</td>
<td>128</td>
<td>14.40</td>
<td><strong>68.09</strong></td>
</tr>
<tr>
<td>Maze (6)</td>
<td>$E_{\min}^M$</td>
<td>143</td>
<td>22.34</td>
<td>71.89</td>
</tr>
<tr>
<td>Maze (10)</td>
<td>$E_{\min}^M$</td>
<td>203</td>
<td>100.21</td>
<td><strong>158.33</strong></td>
</tr>
<tr>
<td>Grid (3)</td>
<td>$E_{\min}^M$</td>
<td>165</td>
<td>2.90</td>
<td>38.94</td>
</tr>
<tr>
<td>Grid (4)</td>
<td>$E_{\min}^M$</td>
<td>381</td>
<td>4.32</td>
<td>79.99</td>
</tr>
<tr>
<td>Grid (5)</td>
<td>$E_{\min}^M$</td>
<td>727</td>
<td>6.623</td>
<td>91.42</td>
</tr>
<tr>
<td>Grid (10)</td>
<td>$E_{\min}^M$</td>
<td>5457</td>
<td><strong>13.630</strong></td>
<td><strong>268.40</strong></td>
</tr>
<tr>
<td>RockSample[4, 4]</td>
<td>$E_{\max}^M$</td>
<td>2432</td>
<td>17.71</td>
<td>35.35</td>
</tr>
<tr>
<td>RockSample[5, 5]</td>
<td>$E_{\max}^M$</td>
<td>8320</td>
<td>18.40</td>
<td><strong>43.74</strong></td>
</tr>
<tr>
<td>RockSample[7, 8]</td>
<td>$E_{\max}^M$</td>
<td>166656</td>
<td>20.32</td>
<td><strong>860.53</strong></td>
</tr>
</tbody>
</table>
Conclusion

• Novel way to generate **provably correct** POMDP strategies
• Good scalability, **not optimal**
• Results **transferrable**
• Future work: **More principled** approach to finite-memory strategies —> Extract FSC directly from RNN