From POMDP executions to policy specifications

Daniele Meli, Giulio Mazzi, Alberto Castellini and Alessandro Farinelli

Department of Computer Science, University of Verona, Italy

Abstract

Partially Observable Markov Decision Processes (POMDPs) allow modeling systems with uncertain state using probability distributions over states (called beliefs). However, in complex domains, POMDP solvers must explore large belief spaces, which is computationally intractable. One solution is to introduce domain knowledge to drive exploration, in the form of logic specifications. However, defining effective specifications may be challenging even for domain experts. We propose an approach based on inductive logic programming to learn specifications with confidence level from observed POMDP executions. We show that the learning approach converges to robust specifications as the number of examples increases.

Keywords

Partially Observable MDPs, Inductive Logic Programming, Answer Set Programming, Explainable AI

1. Introduction

Partially Observable Markov Decision Processes (POMDPs) is a popular framework for modeling systems with state uncertainty [1]. The state cannot be completely observed by the agent, hence it is modeled as a probability distribution called *belief*. A POMDP agent can execute actions, depending on the current belief, and each belief-action pair is mapped to a new belief according to a belief update rule that considers all possible state transitions performed by a *transition map*. Finally, a *reward* is assigned to each state-action pair. The goal of the agent is to maximize the cumulative reward (*return*) by selecting optimal actions (through a *policy* function). Unfortunately, computing optimal policies for POMDPs is complex [2], because different belief realizations are possible at each step. In the field of MDPs and reinforcement learning [3, 4, 5], and recently POMDPs [6, 7, 8], one solution is bounding policy search with logic specifications. However, defining them requires usually unavailable knowledge about the policy.

We propose to learn logic specifications offline from traces (belief-action pairs) of POMDP executions. We express specifications in Answer Set Programming (ASP) [9], a state-of-the-art paradigm for logic planning [10, 11, 12]. We convert the belief to ASP representation, in terms of higher-level domain *features* specified by an expert user, in order to find logic relationships between beliefs and actions. Defining features requires only basic domain knowledge (*e.g.*, relevant domain quantities, used to represent any POMDP task instance). We then exploit Inductive Logic Programming (ILP) [13] to infer ASP rules from feature-action pairs. ILP has

OVERLAY 2022: 4th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, November 28, 2022, Udine, Italy

daniele.meli@univr.it (D. Meli); giulio.mazzi@univr.it (G. Mazzi); alberto.castellini@univr.it (A. Castellini); alessandro.farinelli@univr.it (A. Farinelli)

already been used to enhance interpretability of black-box models [14, 15, 16] and mine robotic task knowledge [17, 18].

We show that state-of-the-art ILASP software [19, 20] is able to automatically learn logic rules representing the policy strategy. These rules are human-readable, since they use human-defined features.

2. Background

2.1. Rocksample

In the rocksample domain, an agent can move in cardinal directions (north, south, east and west) on a $N \times N$ grid, with the goal to reach and sample a set of M rocks with known positions. Rocks may have a good or bad value, yielding to positive or negative reward when sampled, respectively. The observable part of the state is the position of the agent and rocks, while the unobservable part is the value of rocks, modeled with belief distribution. The agent can check the value of rocks to reduce uncertainty. Finally, the agent gets a positive reward exiting the grid from the right-hand side. In this paper, we use $N = 12$ and $M = 4$.

2.2. Answer Set Programming (ASP) and Inductive Logic Programming (ILP)

As of standard syntax [21], an ASP program represents a domain of interest with a *signature* and *axioms*. The signature is the alphabet of the domain, defining *variables* with their ranges (*e.g.*, rock identifiers `R`$\in \{1..4\}$ in rocksample); and *atoms*, *i.e.*, predicates of variables (*e.g.*, actions or environmental features as `dist(R,D)` for representing distance `D` between agent and rock `R`). A variable with assigned value is *ground*, and an atom is ground if its variables are ground. Axioms define logical implications between atoms in the form `h :- b`$_{1..n}$, (*i.e.*, $\bigwedge_{i=1}^{n} \mathtt{b}_i \rightarrow \mathtt{h}$). For instance, in rocksample, axiom `sample(R) :- dist(R,0)` means that a rock can be sampled if the agent is on it. ASP solvers start from known ground atoms (determined from observable state and belief in POMDP traces) and propagate them through axioms to compute a ground program (i.e., with ground atoms). Ground atoms are interpreted as Booleans and true atoms are returned. For instance, in previous example, if `dist(1,0)`, then `sample(1)` becomes executable.

A generic ILP problem $\mathcal{T}$ under the ASP semantics is defined as $\mathcal{T} = \langle B, S_M, E \rangle$, where B is the *background knowledge*, *i.e.*, a set of ASP axioms, atoms and variables; S_M is the *search space*, *i.e.*, the set of candidate ASP axioms to be learned; and E is a set of *examples*, *i.e.*, *context-dependent partial interpretations*, namely couples $\langle e, C \rangle$, being e a *partial interpretation* and C the *context*. A Partial Interpretation (PI) is a pair of sets of ground atoms (actions in this paper) $\langle e^{inc}, e^{exc} \rangle$, being e^{inc} the *included set* and e^{exc} the *excluded set*. The context is a set of ground atoms, here representing domain features. The goal of $\mathcal{T}$ is to find $H \in S_M$ such that e^{inc} can be grounded from $B \cup H \cup C$, while e^{exc} cannot. ILASP [19] finds H which satisfies most examples, also returning the number of counterexamples (*i.e.*, examples whose e^{inc} / e^{exc} is not / is a PI of H).

3. Learning ASP rules from POMDP traces

3.1. ASP representation of the task

The first step of our methodology for learning ASP specifications from POMDP executions is to define a map $F : \mathcal{B} \to G(\mathcal{F})$ from the belief space $\mathcal{B}$ to the set $G(\mathcal{F})$ of possible groundings of $\mathcal{F}$, *i.e.*, the set of user-defined features. Map F thus translates the belief distribution to ground ASP atoms. In rocksample, we define the following features: `guess(R,V)`, *i.e.*, probability `V`$\in \{0, 10, ..., 100\}$ that `R` is valuable; `dist(R,D)`, *i.e.*, the 1-norm `D`$\in \mathbb{N}$ between positions of agent and `R`; `delta_x(R,D)` and `delta_y(R,D)`, *i.e.*, `D`$\in \mathbb{Z}$ is x- or y-coordinate of `R` with respect to agent; bounds on `D,V` (*e.g.*, `D<1`); `sampled(R)` to mark sampled rocks; and `num_sampled(N)`, *i.e.*, `N`$\in \{0, 10, ..., 100\}$ percentage of rocks were sampled.

We represent the set `A` of actions as ASP atoms, *e.g.*, `sample(R)`, `north`. We also introduce atom `target(R)` to identify next rock to sample and capture intention of the agent.

3.2. ILASP problem definition

We are now interested in finding ASP axioms matching features to actions. With reference to the notation of Section 2.2, B contains variables and ranges defined in Section 3.1. S_M is the set of all possible axioms `a :-` $\mathtt{b}_{1..n}$, being `a` an action and $\mathtt{b}_{1..n}$ features. The set E is composed as follows: whenever an action `a` is executed, an example is generated with $e^{inc} = \{\mathtt{a}\}$ and $e^{exc} = \emptyset$; on the contrary, when `a` is not executed, $e^{exc} = \{\mathtt{a}\}$ and $e^{inc} = \emptyset$. The context is computed from belief with map F. In this way, we learn axioms which explain not only why an action was executed, but also cases when it was not.

4. Experimental results

We generate 1000 different rocksample executions with a state-of-the-art planner [22], randomizing positions and values of rocks and initial positions for the agent. We then construct ILASP examples only from executions with returns greater than or equal to the average of all executions in order to learn only from "good" evidence. Overall, 8487 examples for each action are generated. ILASP tasks are run separately for each action for computational efficiency. As an example of learned axioms, the one for `sample(R)` follows:

$$\texttt{sample(R) :- target(R),dist(R,D),D} \leq \texttt{0,not sampled(R),guess(R,V),V} \geq \texttt{90.} \quad (1)$$

meaning that an unsampled rock (`not sampled(R)`) can be sampled when the agent is on it (`dist(R,D), D` $\leq$ `0`) and the rock is valuable with high probability (`guess(R,V),V` $\geq$ `90`). Figure 1 shows the learning results, selecting different percentages of examples in the training set. We want to discover ASP axioms underlying policy computation from patterns in execution traces, hence we do not have access to ground truth specifications to compute standard evaluation metrics and assess learning performance. Instead, we evaluate *percentage of counterexamples* (on the left) and *distance between learned axioms* (on the right) for different number (#) of training examples, with respect to axioms learned from the full dataset. Given 2 rules R_1, R_2, each one made of a set of atoms $\{\mathtt{a}_i\}, i \in \{1, 2\}$, we define distance $R_1 - R_2 =$

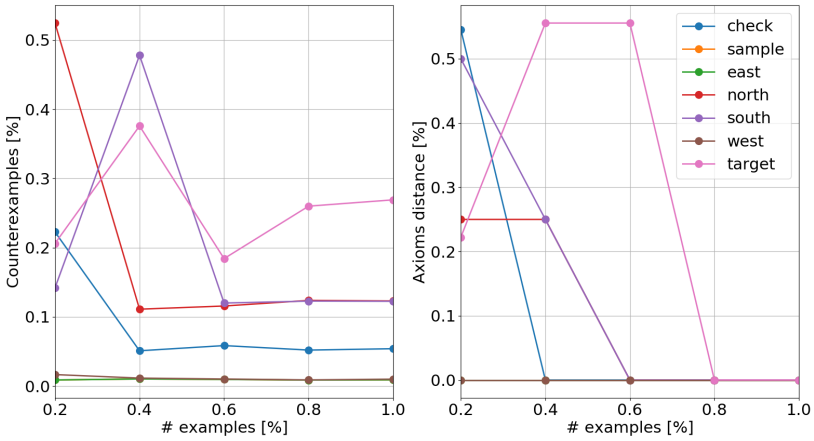

Figure 1: Learning results for different amounts of examples, expressed as percentage of the full dataset.

$|\{a_1\} \cup \{a_2\}| - |\{a_1\} \cap \{a_2\}|$. For instance, `sample(R) :- dist(R,V), V`$\leq 2$ has a distance 5 from (1), due to the missing `not sampled(R)`, `guess(R, V)`, `V`≥ 90 and `target(R)` and the different upper bound on distance D. The chart on the right of Figure 1 reports distances normalized with respect to the number of atoms in final axioms (*i.e.*, axioms using 100% examples). We observe that using $\geq 80\%$ of the dataset, the percentage of counterexamples stabilizes and the distance becomes null for all actions, thus learning successfully converges to a specific hypothesis. Overall, examples learned from the full dataset cover more than 73% of examples.

5. Conclusion

We have proposed a method based on ILP and ASP to induce logic specifications explaining POMDP policies, starting from examples of POMDP executions. Our approach only requires the definition of high-level domain-dependent features from an expert user, which is easier than defining the structure of specifications. Our axioms are enriched with a level of confidence, corresponding to the number of covered examples in the dataset. The confidence level converges as the number of considered examples in the dataset increases, as well as the distance between learned axioms. Furthermore, at least 73% of > 8400 examples are covered, proving the goodness of our axioms. In the future, we aim to include learned specifications in online POMDP solvers to specify new kinds of constraints [23] able to improve performance and efficiency.

Acknowledgments

This project has received funding from the Italian Ministry for University and Research, under the PON “Ricerca e Innovazione” 2014-2020 (grant agreement No. 40-G-14702-1).

References

[1] A. R. Cassandra, M. L. Littman, N. L. Zhang, Incremental pruning: A simple, fast, exact method for partially observable markov decision processes, arXiv preprint arXiv:1302.1525 (2013).

[2] C. H. Papadimitriou, J. N. Tsitsiklis, The complexity of markov decision processes, Mathematics of operations research 12 (1987) 441–450.

[3] M. Leonetti, L. Iocchi, P. Stone, A synthesis of automated planning and reinforcement learning for efficient, robust decision-making, Artificial Intelligence 241 (2016) 103–130.

[4] M. Sridharan, M. Gelfond, S. Zhang, J. Wyatt, Reba: A refinement-based architecture for knowledge representation and reasoning in robotics, Journal of Artificial Intelligence Research 65 (2019) 87–180.

[5] G. De Giacomo, L. Iocchi, M. Favorito, F. Patrizi, Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications, in: Proceedings of the international conference on automated planning and scheduling, volume 29, 2019, pp. 128–136.

[6] G. Mazzi, A. Castellini, A. Farinelli, Identification of unexpected decisions in partially observable monte-carlo planning: A rule-based approach, in: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), International Foundation for Autonomous Agents and Multiagent Systems, 2021, p. 889–897.

[7] G. Mazzi, A. Castellini, A. Farinelli, Rule-based shielding for partially observable monte-carlo planning, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 31, 2021, pp. 243–251.

[8] G. Mazzi, A. Castellini, A. Farinelli, Active generation of logical rules for pomcp shielding, in: Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS '22, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2022, p. 1696–1698.

[9] V. Lifschitz, Answer set planning, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 1999, pp. 373–374.

[10] E. Erdem, V. Patoglu, Applications of asp in robotics, KI-Künstliche Intelligenz 32 (2018) 143–149.

[11] M. Ginesi, D. Meli, A. Roberti, N. Sansonetto, P. Fiorini, Autonomous task planning and situation awareness in robotic surgery, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 3144–3150.

[12] E. Tagliabue, D. Meli, D. Dall'Alba, P. Fiorini, Deliberation in autonomous robotic surgery: a framework for handling anatomical uncertainty, arXiv preprint arXiv:2203.05438, in publication for IEEE ICRA 2022 (2022).

[13] S. Muggleton, Inductive logic programming, New generation computing 8 (1991) 295–318.

[14] J. Rabold, M. Siebers, U. Schmid, Explaining black-box classifiers with ilp–empowering lime with aleph to approximate non-linear decisions with relational rules, in: International Conference on Inductive Logic Programming, Springer, 2018, pp. 105–117.

[15] F. A. D'Asaro, M. Spezialetti, L. Raggioli, S. Rossi, Towards an inductive logic programming approach for explaining black-box preference learning systems, in: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, volume 17, 2020, pp. 855–859.

[16] G. De Giacomo, M. Favorito, L. Iocchi, F. Patrizi, Imitation learning over heterogeneous agents with restraining bolts, in: Proceedings of the international conference on automated planning and scheduling, volume 30, 2020, pp. 517–521.
[17] D. Meli, P. Fiorini, M. Sridharan, Towards inductive learning of surgical task knowledge: A preliminary case study of the peg transfer task, Procedia Computer Science 176 (2020) 440–449.
[18] D. Meli, M. Sridharan, P. Fiorini, Inductive learning of answer set programs for autonomous surgical task planning, Machine Learning 110 (2021) 1739–1763.
[19] M. Law, A. Russo, K. Broda, The ILASP system for learning answer set programs, www.ilasp.com, 2015.
[20] M. Law, A. Russo, K. Broda, Iterative learning of answer set programs from context dependent examples, Theory and Practice of Logic Programming 16 (2016) 834–848.
[21] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, Asp-core-2 input language format, Theory and Practice of Logic Programming 20 (2020) 294–309.
[22] D. Silver, J. Veness, Monte-carlo planning in large pomdps, Advances in neural information processing systems 23 (2010).
[23] A. Castellini, G. Chalkiadakis, A. Farinelli, Influence of State-Variable Constraints on Partially Observable Monte Carlo Planning, in: IJCAI 2019, Macao, China, August 10-16, 2019, ijcai.org, 2019, pp. 5540–5546.