Learning Logic Specifications for Policy Guidance in POMDPs: an Inductive Logic Programming Approach

Daniele Meli   DANIELE.MELI@UNIVR.IT
Alberto Castellini   ALBERTO.CASTELLINI@UNIVR.IT
Alessandro Farinelli   ALESSANDRO.FARINELLI@UNIVR.IT
Department of Computer Science, University of Verona,
strada Le Grazie 15, 37135, Verona, IT

Abstract

Partially Observable Markov Decision Processes (POMDPs) are a powerful framework for planning under uncertainty. They allow to model state uncertainty as a belief probability distribution. Approximate solvers based on Monte Carlo sampling show great success to relax the computational demand and perform online planning. However, scaling to complex realistic domains with many actions and long planning horizons is still a major challenge, and a key point to achieve good performance is guiding the action-selection process with domain-dependent policy heuristics which are tailored for the specific application domain. We propose to learn high-quality heuristics from POMDP traces of executions generated by any solver. We convert the belief-action pairs to a logical semantics, and exploit data- and time-efficient Inductive Logic Programming (ILP) to generate interpretable belief-based policy specifications, which are then used as online heuristics. We evaluate thoroughly our methodology on two notoriously challenging POMDP problems, involving large action spaces and long planning horizons, namely, rocksample and pocman. Considering different state-of-the-art online POMDP solvers, including POMCP, DESPOT and AdaOPS, we show that learned heuristics expressed in Answer Set Programming (ASP) yield performance superior to neural networks and similar to optimal handcrafted task-specific heuristics within lower computational time. Moreover, they well generalize to more challenging scenarios not experienced in the training phase (e.g., increasing rocks and grid size in rocksample, incrementing the size of the map and the aggressivity of ghosts in pocman).

1. Introduction

Partially Observable Markov Decision Processes (POMDPs) are a popular framework for modeling systems with (partial) state uncertainty (Cassandra et al., 1997), due, for instance, to inaccurate perception of autonomous agents (such as robots). The uncertainty over the state is modeled as a probability distribution named *belief*, which can be updated through *observations*. Unfortunately, the partial observability assumption significantly increases the computational effort for computing an optimal *policy*, i.e., the best belief-action mapping, based on the current observations and inner state representation of the agent (Papadimitriou & Tsitsiklis, 1987). In fact, optimality would require exploration of the full belief space, which is impossible, given the continuous belief distribution. This is clearly incompatible with complex real-world scenarios involving large observation and state spaces, and requiring fast decision making. Modern computationally efficient POMDP solvers, as Partially Observable Monte Carlo Planning (POMCP) by Silver and Veness (2010), Determinized

Sparse Partially Observable Tree (DESPOT) by Ye et al. (2017) and their extensions (Sunberg & Kochenderfer, 2018; Wu et al., 2021), mitigate this issue, by adopting a particle filter to discretize the belief distribution and Monte Carlo simulations to compute a nearly optimal policy with high cumulative reward. However, the quality of the computed policy heavily depends on the size of the action space and the length of the planning horizon which, in turn, require many simulations and affect the computational burden. For this reason, the performance of POMDP solvers is dramatically influenced by the availability of task-specific *policy heuristics*, representing approximations of the optimal policy, which are used to focus the exploration on most promising actions, thus reducing the number of required online simulations. Policy heuristics are relations mapping the belief of the agent to actions. However, since they approximate the policy function, which is typically unavailable in (PO)MDPs, a significant effort to design accurate and efficient heuristics is required, even to domain experts.

In this work, we extend the methodology and empirical evaluation first presented by Mazzi et al. (2023b) and propose to learn *human-interpretable policy heuristics* from datasets of POMDP executions, combining basic commonsense domain knowledge and Inductive Logic Programming (ILP, Muggleton, 1991). Specifically, starting from example POMDP executions (generated without heuristics), we use ILP to discover logical specifications mapping actions to a higher-level representation of the belief, based on simple domain-specific human-level concepts (*features*). Such concepts are typically present in the definition of the transition map or the reward for a specific POMDP problem, hence they can be easily provided by a user with basic knowledge of the domain. We use the logical formalism of Answer Set Programming (ASP, Lifschitz, 1999), a state-of-the-art method for representing and reasoning on a planning domain (Tran et al., 2023), to represent learned logical specifications, and use them as policy heuristics in online POMDP solvers. Hence, from now on, we will refer interchangeably to *(policy) specifications* and heuristics.

This work makes the following contributions to the state of the art:

- we show how to learn *interpretable logical belief-dependent policy heuristics* from POMDP executions generated with *any off-the-shelf solver* (in this paper, POMCP), requiring *little training data and time* with respect to state-of-the-art methods based, e.g., on neural networks for policy learning;

- we propose a method to integrate learned logical heuristics in several POMDP solvers, mainly POMCP, DESPOT and their most recent and efficient versions, AdaOPS by Wu et al. (2021) and POMCPOW by Sunberg and Kochenderfer (2018). Our approach can take into account the *confidence level* about learned specifications, depending on their adherence to the training data (or alternatively specified by the user, thanks to the interpretable formalism). This is used for *soft policy guidance*, namely, weighted probabilistic action exploration with completeness guarantees, in POMCP, and for selecting the most probable action in DESPOT and AdaOPS;

- we empirically demonstrate our approach in two notably challenging POMDP benchmark domains, namely *rocksample* and *pocman*. These two domains present unique challenges for our methodology, compared to other available POMDP benchmarks. In fact, rocksample is typically the domain with the largest (discrete) action space (Wu

et al., 2021), while pocman (Ye et al., 2017) requires the longest planning horizon (i.e., the longest action sequence required to complete the task). Hence, in such scenarios the quality of policy heuristics is crucial to achieve good performance. We assess the *generalization and scalability capabilities of learned specifications*, which are derived from small domains (e.g., rocksample on a small grid and with few rocks) and then applied to online POMDP planning in more challenging contexts (larger maps and belief spaces). We also perform an ablation study to investigate the role of specifications in different parts of Monte Carlo tree search, i.e., rollout and tree exploration. Finally, we discuss the role of the quality and amount of training data on learned policy heuristics and resulting planning performance;

- we make publicly available the code for experimental replication at `https://gitlab.com/dan11694/ilasp_pomdp`.

Differently from our preliminary paper (Mazzi et al., 2023b), we introduce heuristics also in the most computationally demanding rollout phase of Monte Carlo tree search; we integrate our methodology with the DESPOT solver, while still learning from POMCP-generated traces, showing that training data can be derived from any POMDP solver; we replace the custom battery domain with a more challenging and famous benchmark, pocman, which presents much longer planning horizon (up to ≈ 85 steps to complete the task); we make a more extensive empirical evaluation of our approach, including a more detailed discussion of the potential limitations of our method, and the comparison with other state-of-the-art POMDP solvers POMCPOW and AdaOPS, and neural architectures.

In Section 2 we review recent research in heuristics learning and the fusion of (PO)MDP and logics, highlighting the differences with respect to our approach. Then, in Section 3 we introduce relevant background and notation about POMDP, ASP and ILP, with a description of our benchmark domains as useful leading examples through the rest of the paper. We then present our methodology in Section 4, and evaluate it thoroughly in Section 5. We finally discuss advantages and potential limitations of our approach in Section 6, and summarize our results and possible future extensions in Section 7.

2. Related Works

Exact computation of optimal POMDP policies is known to be PSPACE-complete (Papadimitriou & Tsitsiklis, 1987), hence hardly scalable to large action, state and observation spaces. This has often limited the practical application of POMDP solving algorithms to complex real-world scenarios, such as industrial automation, autonomous robotic navigation and medical diagnosis, which would actually benefit from the representational features of this approach (Cassandra, 1998). For this reason, in recent years several *online* POMDP solvers have been proposed, which yield sufficiently reduced computational time for online deployment. Some of them exploit factorization of the state space (Paquet et al., 2005) or task-based reduction (Shani, 2013), in order to reduce the search space and mitigate the computational burden. Currently, the most popular approach relies on Monte Carlo Tree Search (MCTS), which computes an estimate of the value of actions and states with multiple efficient simulations. In this way, the state-action search space is explored only locally (following simulations performed from the current state of the agent), thus mitigat-

ing the curse of dimensionality. MCTS represents the foundation of two state-of-the-art online POMDP solvers, POMCP (Silver & Veness, 2010) and DESPOT (Ye et al., 2017), and their more efficient extensions AdaOPS (Wu et al., 2021) and POMCPOW (Sunberg & Kochenderfer, 2018), which have been employed for challenging tasks, such as active visual search (Wang et al., 2021a; Giuliari et al., 2021), robotic navigation (Gupta et al., 2022) and multi-agent planning by Bhattacharya et al. (2021) and Amini et al. (2023). We will discuss details of POMCP and DESPOT in the following sections, as they represent the algorithmic foundation for more advanced solvers.

Despite the success of online POMDP solvers even in real-world scenarios, their performance degrades significantly in complex environments with very large action and state spaces, or in tasks with long planning horizons, requiring many deep simulations. Indeed, recent literature shows that integrating available knowledge into (PO)MDPs and Reinforcement Learning (RL), can significantly support the computation of efficient policies (Castellini et al., 2019; Zuccotto et al., 2022b, 2022a; Vinyals et al., 2019; Cheng et al., 2021). Such knowledge is either available in advance or, more conveniently, it can be learned as in Zuccotto et al. (2022b). Existing solutions to incorporate domain-specific knowledge into POMDPs span from the use of neural networks and deep learning methodologies, to more classical graph and logical representations. For instance, Zuccotto et al. (2022b) proposed to learn state variable relationships in POMCP using Markov random fields, for application to robotic search. Cai and Hsu (2022) proposed to combine DESPOT with heuristics learned offline as a neural network, and refined online from experience, for autonomous driving in crowds. In the same scenario, Danesh et al. (2023) proposed an actor-critic architecture to learn the value function for MCTS, inspired by the success of AlphaZero and its variants (Silver et al., 2018). A similar approach is used by Lee et al. (2021) to learn macro-actions for DESPOT, i.e., local trajectories in the action space. In the more general area of learning generalizable policy heuristics from experience, De la Rosa et al. (2011) proposed an approach based on binary decision trees, which however offers limited representational capabilities. Toyer et al. (2020) relied on a neural network architecture to learn the heuristics, but assuming that the optimal behavior (i.e., the heuristics themselves) was actually encoded in the training dataset. Recently, Subramanian et al. (2022) proposed an approach based on recurrent neural networks to learn efficient policies for POMDPs, particularly for rocksample, exploiting approximate information maximization and outperforming state-of-the-art neural architectures in the field, based on proximal policy optimization (Schulman et al., 2017).

Methods based on neural networks require many examples to learn from, and a significant amount of time and hardware capabilities (Cai & Hsu, 2022). Even when online learning is proposed, many observation points must be acquired before discovering robust and efficient policy heuristics (e.g., $> 10^5$ iterations in Lee et al., 2021). Moreover, neural networks generate hardly interpretable heuristics, thus providing scarce guarantees about the learned behavior. This may result in exploring bad policies, in case training information is not correct or good enough (as highlighted, e.g., by Cai & Hsu, 2022). A more interpretable knowledge-based approach involves the use of finite state controllers for POMDPs (Andriushchenko et al., 2022), which are however learned online assuming the availability of oracles setting suitable bounds to exploration. Maliah et al. (2022) define contingent planning sub-routines based on a logical formalization of POMDP actions, environmental

features and the goal, in order to support the decision making process under uncertainty. In general, symbolic and logical formalisms have the potential to increase the interpretability of AI systems, enhancing their diffusion and acceptance (Kambhampati et al., 2022). Logical formalisms can also be applied to improve policy computation, though existing works are mainly related to the field of fully observable MDPs and semi-MDPs. For instance, in the REBA framework by Sridharan et al. (2019) ASP is used to describe spatial relations between objects and rooms in a houseware domain, hence driving a robotic agent to choose a specific room to inspect, while solving simpler MDP problems locally. Similarly, Leonetti et al. (2016) propose DARLING, which uses ASP statements to bound MDP exploration in a simulated grid domain (similar to rocksample) and in a real service robotics scenario. Linear temporal logic is used by De Giacomo et al. (2019), Leonetti et al. (2012) to drive exploration in MDPs. Logical statements can help also avoid unwanted behaviors, e.g., in safety-critic scenarios. To this aim, Mazzi et al. (2021a, 2021b, 2023a) refined rule templates to identify unexpected decisions and shield from unwanted actions, in the context of velocity regulation for a mobile robot. In Wang et al. (2021b), a goal-constrained belief space containing only safely reachable states is defined with propositional logics, and a Satisfiability Modulo Theory (SMT) solver is used to guarantee proper execution of an houseware automation task.

The aforementioned approaches however assume that logical statements are defined by the user. This is often infeasible even in relatively simple (PO)MDP domains, since logical heuristics express knowledge related to the policy, which is unknown. Moreover, the wrong definition of logical relations may significantly impact on task performance. For instance, De Giacomo et al. (2020) propose to learn temporal logic specifications as finite automata from good example traces. However, rules are used to shape the reward in fully observable MDPs, hence the method strongly relies on the quality of training examples and induced formulas.

In this paper, we propose to overcome some of existing limitations, learning logical policy heuristics from relatively few examples of POMDP execution (i.e., sequences of belief-action pairs), and using them to drive MCTS exploration, rather then define reward signals or hard constraints having a huge impact on planning performance. We will show that this allows to mitigate the impact of bad policy heuristics, especially in POMCP. We choose the ASP formalism to represent policy heuristics. ASP is the state of the art for logical task planning (Erdem & Patoglu, 2018; Ginesi et al., 2020; Meli et al., 2021b; Tagliabue et al., 2022; Tran et al., 2023; Meli et al., 2023), providing significantly richer syntax than other approaches, such as finite state controllers adopted by Andriushchenko et al. (2022). Several solutions exist to learn domain knowledge expressed in ASP or other logical formalisms, e.g., inductive refinement of action theories and operators proposed by Gil (1994), Balduccini (2007) and relational reinforcement learning by Sridharan and Meadows (2018), Mota and Sridharan (2020). In this paper, we adopt the framework of Inductive Logic Programming (ILP, Muggleton, 1991), specifically Inductive Learning of Answer Set Programs (ILASP, Law et al., 2018). It has been recently used, e.g., to enhance comprehensibility of black-box models (D'Asaro et al., 2020; Rabold et al., 2018), including moral value systems for autonomous agency (Veronese et al., 2023) and learn robotic task knowledge (Meli et al., 2020; Bonet & Geffner, 2020; Rodriguez et al., 2021; Meli et al., 2021a). Recently, ILASP was also used to learn reward machines in online reinforcement learning by Furelos-Blanco

et al. (2021). In contrast to it, we here focus on offline heuristics discovery, rather than learning reward representations. Furthermore, our heuristics represent the belief-action mapping in POMDPs, hence they are based directly on the uncertain belief representation, rather than on observable states.

As evidenced in the following of this paper, ILASP offers considerable advantages with respect to other methods for policy heuristics learning. In fact, it requires significantly fewer examples (< 1000 traces of execution) and less training time (few minutes), with respect to approaches based on neural networks. Furthermore, it can easily incorporate commonsense domain knowledge, e.g., task-specific concepts, which allow to learn interpretable heuristics. As we will show in the next sections, commonsense concepts encode very simple information which can be directly retrieved from the POMDP problem definition itself, thus requiring minimal expertise. Another important advantage of our approach is that ILASP can discover high-quality generalizable heuristics even when the training examples are obtained in small-scale scenarios by a solver unscented of the optimal heuristics, differently from, e.g., De la Rosa et al. (2011), Toyer et al. (2020).

3. Background

In this section, we provide basic description and notation about the foundational parts of our methodology. Specifically, we first introduce the general POMDP framework and describe the rocksample and pocman domains, which are two benchmark and notably challenging domains for the POMDP community (Ye et al., 2017; Wu et al., 2021). Then, we describe POMCP and DESPOT algorithms for online POMDP solving, on which more advanced solvers as AdaOPS and POMCPOW are based. Finally, we present ASP and ILASP, with clarifying examples from the rocksample domain.

3.1 Partially Observable Markov Decision Processes

A Partially Observable Markov Decision Process (POMDP, Kaelbling et al., 1998) is a tuple $(S, A, O, T, Z, R, \gamma)$, where S is a set of partially observable *states*; A is a set of *actions*; Z is a set of *observations*; $T \colon S \times A \to \Pi(S)$ is the *state-transition model*, mapping to a probability distribution $\Pi(\cdot)$ over states; $O \colon S \times A \to \Pi(Z)$ is the *observation model*; R is the *reward function* and $\gamma \in [0, 1)$ is a *discount factor*. An agent must maximize the *discounted return* $E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$. The probability distribution over states $\mathcal{B} = \Pi(S)$, called *belief*, is used to model uncertainty about the true state. To solve a POMDP it is required to find a *policy*, namely a function $\pi \colon \mathcal{B} \to A$ that maps beliefs $\mathcal{B}$ into actions. However, exact computation of the optimal policy requires representation of the full belief-action space, i.e., exploration of the full continuous belief distribution and evaluation of all possible actions under any circumstances. For this reason, exact solving of POMDPs is PSPACE-complete (Papadimitriou & Tsitsiklis, 1987), and impractical even in relatively small domains of interest, as the benchmark ones presented hereafter.

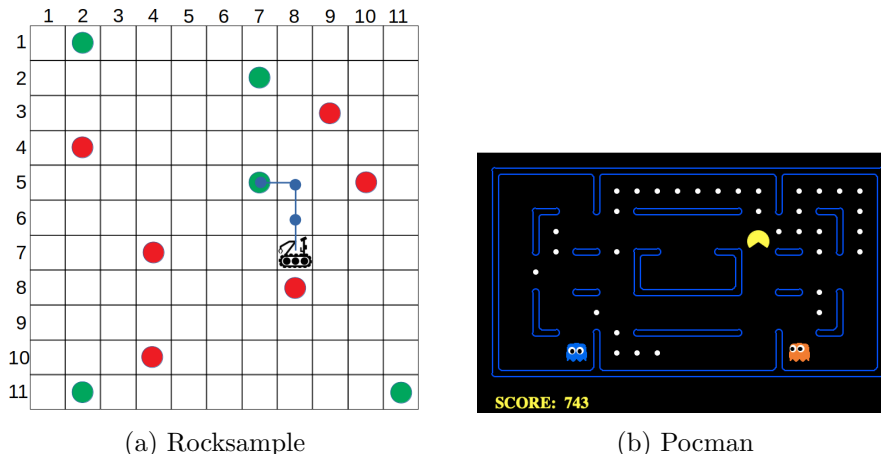We now show how our benchmark domains are represented as POMDPs.

(a) Rocksample   (b) Pocman

Figure 1: Example scenarios for our two case studies.

3.1.1 ROCKSAMPLE SCENARIO

In the rocksample domain, an agent can move in cardinal directions on a $N \times N$ grid, one cell at a time, with the goal to reach and sample a set of M rocks with known position on the grid. Rocks may be valuable or not. Sampling a valuable rock yields a positive reward (+10), while sampling a worthless rock yields negative reward (-10).

The observable state of the system is described by the current position of the agent $p_c = (x_c, y_c)$ and each i-th rock $p_i = (x_i, y_i)$, while the value of rocks is uncertain before sampling. However, the agent can check one rock per time step and obtain an observation about its value, with an accuracy depending on the distance between the rock and the agent. Finally, the agent obtains a positive reward (+10) exiting the grid from the right-hand side. The belief then contains $\{Pr(r_i = 1)\}_{i=1...M}$, i.e., the probability that each rock r_i is valuable ($r_i = 1$).

An explanatory scenario with $N = 11$ and $M = 11$ is depicted in Figure 1, where valuable and invaluable rocks are marked as green and red dots, respectively, and the agent is at location $(8, 7)$ initially. The rocksample task is particularly challenging because of the large number of actions. In fact, $\mathcal{A} = \{\texttt{sample}, \texttt{exit}, \texttt{north}, \texttt{south}, \texttt{west}, \texttt{east}, \texttt{check}(i) \forall i = 1 \ldots M\}$, hence, $|\mathcal{A}| = 6 + M$. Moreover, the belief size increases with M as well, while the state space depends also on the size N of the map.

3.1.2 POCMAN SCENARIO

In the pocman domain, an agent (pocman) can move in cardinal direction (hence, 4 actions) in a grid world with N cells, possibly presenting walls, with the goal to eat pellets of food, yielding a positive reward (+1). Each cell contains food pellets with probability ρ_f. G ghost agents are also present in the grid, which normally move randomly, but may start chasing the pocman (with probability ρ_g) if they are within 2-cell distance from it. If the pocman clears the level, i.e., it eats all food pellets, a positive reward is yielded (+1000). If the pocman is eaten or hits a wall, it receives a negative reward (-100). Moreover, at each step the pocman gets a small negative reward (-1).

The fully observable state contains coordinates of each i-th wall cell $p_{wi} = (x_{wi}, y_{wi})$ and the current position of pocman $p_c = (x_c, y_c)$. At each step, the pocman receives *certain information* (observation) about surrounding environment within 2-cell distance. Specifically, within this range the agent can see or hear ghosts and smell food. Then, the agent builds two belief distributions, $\mathcal{B}_g = \{Pr(g_i = 1)\}_{i=1...N}, \mathcal{B}_f = \{Pr(f_i = 1)\}_{i=1...N}$, representing the probability that i-th cell contains a ghost ($g_i = 1$) or a food pellet ($f_i = 1$), respectively.

An explanatory scenario with $G = 2$ is depicted in Figure 1b. Pocman is a particularly challenging task because of the very large dimension of the state and belief space ($\approx 10^{56}$ states, as reported by Ye et al., 2017) and the long planning horizon (to clear the level, the agent must potentially explore the full environment to find all pellets, and spurious movements are often required to escape ghosts). Given the step penalization for pocman, it is then fundamental to optimize action selection to achieve good planning performance.

3.2 Online Solvers for POMDPs

We now describe the two main online POMDP solvers on which our methodology relies: POMCP and DESPOT.

3.2.1 PARTIALLY OBSERVABLE MONTE CARLO PLANNING (POMCP)

POMCP (Silver & Veness, 2010) computes a sub-optimal policy based on Monte Carlo techniques. The belief distribution is approximated using a *particle filter*. More specifically, each time the agent receives an observation, it perceives a specific state realization s_i. A particle is then associated to each s_i, and the probability of each realization is $\frac{p_i}{\sum_{i=1}^{|S|} p_i}$, being p_i the number of particles corresponding to s_i. In other words, the continuous belief distribution is approximated as a set of particles partitioned between multiple state realizations. The particle filter can be initialized randomly or considering prior knowledge about the environment, as shown by Castellini et al. (2019).

POMCP then computes an online policy, i.e., the best action to be performed at each time step, combining the particle filter with MCTS. Specifically, during execution, the history h of explored belief-action pairs is stored. At each step, a tree is built from the current state (root), branching across all possible actions. Then, multiple simulations are performed, i.e., multiple actions are chosen, resulting in exploration of new future states, approximated via the particle filter. A particle (representing a specific state) is selected from the filter and used as an initial point to perform a simulation in the Monte Carlo tree. In order to reduce the number of simulations and exploit the particle filter efficiently, Upper Confidence Bound for Trees (UCT, Kocsis & Szepesvári, 2006) is used as a search strategy to select most convenient subtrees to explore. In particular, UCT suggests to explore action a which maximizes the action value:

$$V_{UCT}(ha) = V(ha) + c \cdot \sqrt{\frac{\log N(h)}{N(ha)}} \tag{1}$$

where $V(ha)$ is the expected return achieved by selecting action a, $N(h)$ is the number of simulations performed from history h, $N(ha)$ is the number of simulations performed

while selecting action a, and c is known as the exploration constant. UCT is used to balance the exploration of new actions (i.e., actions with low $N(ha)$) and the exploitation of effective actions (i.e., action with high $V(ha)$). In case a leaf node is reached in the tree (i.e., $N(ha) = 0$), an action is chosen according to a default *rollout* criterion (typically, randomly). At the end of all simulations for a given step, only the action which leads to the highest discounted return is executed.

POMCP has the advantage to use simulations to explore only a specific part of the belief-action space, relevant from the current state of execution. However, it fails to scale to very large belief spaces, or when many actions are available, since it would require a large amount of simulations in order to properly estimate the value of actions and closely approximate the optimal policy. For this reason, in Silver and Veness (2010) heuristics are still needed to achieve good performance in challenging scenarios as pocman and rocksample. Other solutions have also been proposed to mitigate the computational burden, e.g., by leveraging on model-specific structural properties (Katt et al., 2017) or existing state variable relationships to shrink the search space and require fewer simulations (Castellini et al., 2019; Zuccotto et al., 2022b), and parallel solving (Basu et al., 2021).

3.2.2 DETERMINISTIC SPARSE PARTIALLY OBSERVABLE TREES (DESPOT)

DESPOT (Ye et al., 2017) tries to solve some POMCP limitations, by an intensive use of heuristics to drive the exploration in MCTS. Assuming very good heuristics are available, relatively few but carefully selected *scenarios*, i.e., Monte Carlo sub-trees, are explored by the algorithm and allow to converge to a nearly optimal policy.

In more detail, starting from a root belief node b_0 (represented as a set of particles), DESPOT approximates the belief-action tree in simulation by K random scenarios, consisting of particle-observation-action sequences (each particle represents a specific state) in the form $\langle s_0, a_0, s_1, o_1, a_1, \ldots, s_n, o_n, a_n \rangle$, being s_i a specific particle at a given belief node. Similarly to POMCP, the goal of DESPOT is to compute an optimal policy by building and exploring an approximate belief-action tree, executing a *default policy* (equivalent to the rollout policy) if observations out of the K scenarios are received. Given a policy π, the algorithm estimates the value function $V_\pi(b_0)$ resulting at b_0 under π as:

$$\hat{V}_\pi(b_0) = \frac{1}{K} \sum_{i=1}^{K} V_{\pi,\Phi_i}$$

being Φ_i the i-th scenario and V_{π,Φ_i} the discounted return achieved from π in Φ_i. Clearly, $\hat{V}_\pi(b_0) \to V_\pi(b_0)$ as $K \to \infty$, which however is in conflict with online application. On the other hand, choosing a finite (relatively low) value for K induces the risk of *overfitting* the policy on a limited set of scenarios. In order to mitigate this issue, DESPOT performs *regularization* on the size of the optimal policy, i.e., it maximizes $\hat{V}_\pi(b_0) - \lambda|\pi|$, with $\lambda \in \mathbb{R}^+$ and $|\cdot|$ denoting the length of π, i.e., the number of actions to be executed.

The key extension of DESPOT with respect to POMCP is the use of pre-defined lower and upper bounds on $\hat{V}_\pi(b_0)$, respectively $l(b_0)$ and $u(b_0)$, in order to build an useful and informative belief-action sub-tree and perform efficient *anytime heuristics search*. The lower bound is computed as:

$$l(b_0) = \hat{V}_{\pi_0}(b_0) \tag{2}$$

being π_0 the pre-defined *default policy* used at leaf nodes. In general, given $\epsilon(b) = u(b) - l(b)$, the goal of DESPOT is to incrementally reduce $\epsilon(b_0)$ to $\xi\epsilon(b_0), 0 < \xi < 1$, until $\epsilon(b_0) < \epsilon_0 \in \mathbb{R}^+$. In fact, this means that the lower and upper bound become more and more similar, thus the algorithm is *more aware* about the current state and more confidently performs the best action. More specifically, at each root node, the algorithm uses the upper bound to select an action $a^\star$ to be explored, as:

$$a^\star = \operatorname*{argmax}_{a \in A} u(b_0, a) = \operatorname*{argmax}_{a \in A} \left\{ \rho(b_0, a) + \sum_{b'} u(b') \right\} \tag{3}$$

being $\rho(b_0, a)$ the average (over all scenarios) discounted return achieved from b_0 executing a, and b' any descendant node generated from b_0 by executing a. Then, the observation branch to be explored is chosen in order to maximize:

$$E(b') = \epsilon(b') - \frac{|\Phi(b')|}{K} \xi\epsilon(b_0) \tag{4}$$

where $|\Phi(b')|$ is the number of scenarios passing through b'. $E(b')$ represents the discrepancy between the current gap at b' and the expected gap resulting from the reduction $\epsilon(b_0) \to \xi\epsilon(b_0)$. As proved by Ye et al. (2017), this allows to actually converge faster.

As stated in the beginning of this section, one crucial requirement of DESPOT is the availability of good task-specific heuristics, including the default policy for the definition of the lower bound. This is a major limitation in complex real-world scenarios, where often domain knowledge is scarcely available.

3.3 Answer Set Programming

An ASP program represents a domain of interest with a *signature* and *axioms* (Calimeri et al., 2020). The signature is the alphabet of the domain, defining its relevant attributes as variables (with discrete ranges) and predicates of variables (*atoms*). For example, in the rocksample domain variables of interest are: rock identifiers $R \in \{1, \ldots, M\}$, possible values of distances between agent and rocks $D \in \mathbb{Z}$, and (discretized) probabilities computed from the belief $V \in \{0, 1, \ldots, 100\}$[1]. Atoms typically represent environmental features and actions. In rocksample, possible environmental features are `guess(R,V)`, representing the discrete (percentage) probability `V` that rock `R` is valuable, and `dist(R,D)`, representing the Manhattan distance `D` between the agent and rock `R`; possible action atoms are `sample(R)` and `east`, representing the action of sampling rock `R` and moving east, respectively.

In ASP, a variable whose value is assigned is said to be *ground* (e.g., R=1). An atom is ground if its variables are ground (e.g., `guess(1, 90)`, meaning that rock 1 is valuable with 90% probability). In particular, we define the *Herbrand base* of an ASP program P (or a set of ASP atoms), $\mathcal{H}_b(P)$, as the set of ground atoms which can be derived from P (in other words, all possible variable assignments for atoms in P). Axioms are logical relations between atoms. In this paper, we consider *normal rules*, *aggregate rules* and *weak constraints*. A *normal rule* `h :- b`$_1$`, ..., b`$_n$ defines preconditions for grounding *head* `h` as the logical conjunction $\bigwedge_{i=1}^{n} b_i$ (*body* of the rule). For instance,

```
sample(R) :- guess(R,V), V>60.
```
(5)

---

1. Since probabilities are normalized in $[0, 1]$, discrete probability values represent percentages.

means that rock R can be sampled if the agent believes it is valuable with probability V> 60%. Symbol :- represents the logical implication ($\leftarrow$) in ASP.

An *aggregate rule* expresses a choice over normal rules. It has the form $\mathtt{l}$ {$\mathtt{h}_j$: $\mathtt{b}_{ij}$, ..., $\mathtt{b}_{nj}$ } $\mathtt{u}$, where symbol : is equivalent to :- and $\mathtt{l},\mathtt{u} \in \mathbb{N}$, $\mathtt{l} \leq \mathtt{u}$. Given the set H of ground heads $\mathtt{h}_j$, according to respective body atoms $\mathtt{b}_{ij}$, the aggregate specifies that only a subset $H_l^u \subseteq H$ can be grounded, such that $\mathtt{l} \leq |H_l^u| \leq \mathtt{u}$. For instance, if the following holds:

$$0 \ \{\texttt{check(R) : guess(R,V), V>60}\} \ 1. \tag{6}$$

and $\texttt{guess(0, 100)}$ and $\texttt{guess(1,90)}$ are ground atoms, then $H = \{\texttt{check(0), check(1)}\}$, but only three subsets of it can be actually grounded, namely $H_0 = \{\}$, $H_1 = \{\texttt{check(0)}\}$, $H_2 = \{\texttt{check(1)}\}$.

A specific choice among the possible ground sets resulting from an aggregate rule can be made following a *weak constraint* in the form:

$$\texttt{:} \sim \mathtt{b}_1(\mathtt{V}_1,\ldots,\mathtt{V}_n),\ldots,\mathtt{b}_m(\mathtt{V}_1,\ldots,\mathtt{V}_n).[\mathtt{w@l}, \ \mathtt{V}_1,\ldots,\mathtt{V}_n]$$

where $\mathtt{w}$ is the weight, $\mathtt{l}$ an integer priority level (used if multiple weak constraints are specified), $\mathtt{b}_i$ are atoms and $\mathtt{V}_j$ variables. The weight can either be one variable among $\mathtt{V}_j$'s, or an integer. If the weight is a positive (respectively, negative) integer, the weak constraint means that grounding of atoms $\mathtt{b}_i$'s should be penalized (respectively, encouraged). If the weight is a variable, say $\mathtt{V}_1$, then the constraint means that lower values for $\mathtt{V}_1$ should be preferred. As an example, if axiom (6) holds and the following weak constraint is specified:

$$\texttt{:} \sim \texttt{check(R), guess(R,V). [-V@1, R, V]} \tag{7}$$

$H_0 \leq H_2 \leq H_1$ will be the preference ordering among above allowed sets.

Given an ASP task description, an ASP solver computes all its *answer sets*. An answer set is the minimal set of ground atoms satisfying axioms. Starting from an initial variable assignment, all ground atoms in the body of axioms are computed, hence ground head atoms are derived. For instance, assuming an ASP program contains only axiom (5) and the set of ground atoms $\{\texttt{guess(1,50), guess(2,70)}\}$, then action $\texttt{sample(2)}$ will be grounded by the solver, and the unique answer set will contain all three ground atoms, i.e., $\{\texttt{guess(1,50), guess(2,70), sample(2)}\}$.

3.4 Inductive Logic Programming under the Answer Set Semantics

An ILP problem $\mathcal{T}$ under the answer set semantics is defined as the tuple $\mathcal{T} = \langle B, S_M, E \rangle$, where B is the *background knowledge*, i.e. a set of atoms and axioms in ASP syntax (e.g., ranges of variables); S_M is the *search space*, i.e. the set containing all possible ASP axioms that can be learned; and E is a set of *examples*, i.e., a set of ground atoms (constructed, e.g., from traces of execution). Informally, the goal is to find a *hypothesis*, i.e., a set $H \in S_M$ that explains as many as possible of the examples in E. To this end, we use state-of-the-art ILASP learner (Law et al., 2018), where examples are *Context-Dependent Partial Interpretations* (CDPIs).

Definition 1 (Partial interpretation). Let P be an ASP program. A *partial interpretation* of P is defined as $e = \langle e^{inc}, e^{exc} \rangle$, where e^{inc} is named *included set*, i.e., a subset of ground atoms which can be part of an answer set of P; e^{exc} is named *excluded set*, i.e., a subset of ground atoms which are not part of an answer set of P.

Definition 2 (Context-dependent partial interpretation (CDPI)). A CDPI of an ASP program P is a tuple $\langle e, C \rangle$, where e is a partial interpretation of P and C is a set of atoms called *context*.

In this paper, partial interpretations contain atoms for actions, while the context involves environmental atoms. In this way, policy specifications (representative of the belief-action map) can be learned.

We can now formalize the ILASP problem considered in this paper, hence the properties to be satisfied by H:

Definition 3 (ILASP task with CDPIs). An ILASP learning task with CDPIs is a tuple $\mathcal{T} = \langle B, S_M, E \rangle$, where E is a set of CDPIs such that:

$$\forall e = \langle \langle e^{inc}, e^{exc} \rangle, C \rangle \in E : \ B \cup H \cup C \models e^{inc} \wedge \ B \cup H \cup C \not\models e^{exc}$$

In other words, ILASP finds axioms which guarantee that actions in e^{inc}, observed in the examples, can be executed (i.e., can be grounded in an answer set), while unobserved actions in e^{exc} cannot, given the context set of environmental features[2]. In addition, ILASP finds the *minimal* hypothesis H, i.e., axioms with the least number of atoms explaining examples. This may contrast with the accuracy of specifications, but increases their interpretability. Moreover, ILASP algorithm (Law, 2023) solves the learning problem as an instance of conflict-driven constraint learning (Marques-Silva et al., 2021), which tries to balance between these two factors. ILASP can also learn weak constraints from *ordered CDPIs* (Law et al., 2016), i.e., partial interpretations with pre-defined preference values pr, expressed in the form $\langle \langle e^{inc}, e^{exc} \rangle, C \rangle @pr$.

Finally, ILASP finds the hypothesis which explains *most* of the examples, i.e., there may be CDPIs which are not entailed by $H \cup B$. When a hypothesis is found, ILASP also returns the number of not covered CDPIs, which can be used to quantify the level of confidence of H with respect to a given example set.

4. Methodology

We now describe our methodology for learning policy-related logical (ASP) specifications from POMDP traces of execution. Our goal is to find interpretable policy heuristics encoded in the traces of execution, which consist of belief-action pairs. As an example, in the rocksample domain, the handcrafted policy heuristics by Silver and Veness (2010) is:

---

2. This section, and specifically Definition 3, ignores the concept of *negative examples* in ILP (Law et al., 2018), i.e., examples where e^{inc} contains a set of *forbidden* ground atoms. In fact, in this paper we extract examples from POMDP executions, hence only from *observed behavior*. A negative example, on the contrary, allows to learn *hard constraints*, i.e., conditions that shall never be verified. Typically, negative examples are provided by domain experts, hence they are not considered for the scope of this work.
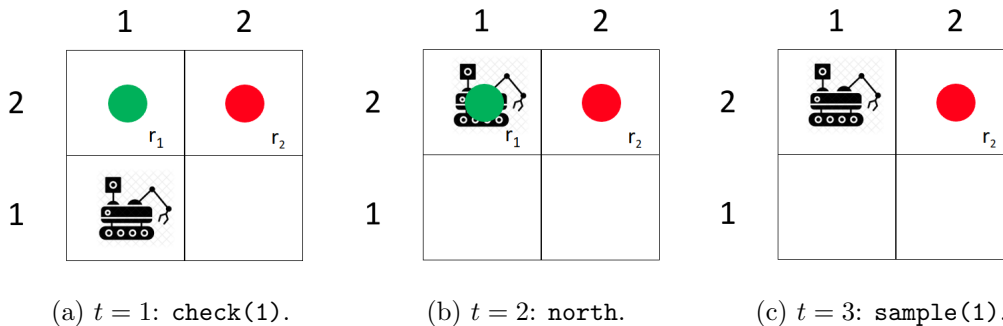
(a) $t = 1$: `check(1)`. (b) $t = 2$: `north`. (c) $t = 3$: `sample(1)`.

Figure 2: Simplified rocksample scenario with $N = M = 2$.

> *Either check a rock whenever bad observations were received and it has been measured few times (< 5); or sample a rock if the agent is at its location and collected observations are more positive (good value) than bad; or move towards a rock with more good than bad observations.*

It relates the execution of specific actions to the *observations* received by the agent. On the contrary, we want to discover policy heuristics *depending on the belief distribution*. To this aim, three steps are required:

- ASP formalization of the POMDP problem;

- generation of ILASP examples from execution traces;

- definition of the ILASP task.

In the following, we explain the phases of our pipeline, with reference to the clarifying example of a simple rocksample domain in a 2×2 ($N = 2$) grid with $M = 2$ rocks (Figure 2). Finally, we show how to use learned policy specifications for online POMDP solving in POMCP and DESPOT.

4.1 ASP Formalization of the POMDP Problem

The POMDP problem is originally formalized as in Section 3.1. In order to represent it in ASP formalism, we need to define a set of high-level environmental features, which represent basic concepts about the belief. We denote the set of environmental features as $\mathcal{F} = \{\mathtt{F}_i\}$. In the rocksample domain, such concepts depend on the variables introduced in Section 3.3 and include[3]:

- `guess(R,V)`, representing the probability `V` that rock `R` is valuable;

- `dist(R,D)`, representing the Manhattan distance `D` between the agent and rock `R`;

---

3. With respect to the work presented by Mazzi et al. (2023b), we omit `min_dist(R)`, `best_guess(R)`, representing higher-level concepts about the task which cannot be directly retrieved from the basic POMDP definition. In this way, we learn slightly different policy specifications, which are efficient for online planning and make our approach more generalizable to other domains.

- `delta_x(R,D)` and `delta_y(R,D)`, representing the relative position `D` along x and y axis of rock `R` with respect to the agent;

- `sampled(R)`, meaning that rock `R` has been sampled;

- `num_sampled(V)`, meaning that a percentage `V` of rocks has been sampled.

As stated at the beginning of this paper, environmental atoms represent simple domain concepts which appear in the definition of the transition and reward maps for the POMDP problem, hence they can be easily defined by the user. For instance, `sampled(R)` is the consequence of the sampling action on rock `R`, while `guess(R,V)` is updated after the agent senses (checks) rock `R`. Similarly, we must define the set of ASP atoms $\mathcal{A} = \{act_i\}$ representing the action space. In the rocksample domain, they are `sample(R)`, `check(R)`, `east`, `north`, `south`, `west`, with straightforward interpretation.

4.2 Generation of ILASP Examples

Given the ASP formalization of the problem, we can represent the traces of execution accordingly, in order to extract examples (i.e., CDPIs) for ILASP. Traces of execution are sequences of belief-action pairs. Hence, we need to lift POMDP belief and actions to the equivalent ASP representation. Specifically, the belief is lifted using the *feature map* $F_\mathcal{F} : \mathcal{B} \to \mathcal{H}_b(\mathcal{F})$. For instance, in the simplified rocksample scenario of Figure 2a, assume that 100 particles are available to approximate the belief. At the initial time $t = 1$ of the task, particles are distributed uniformly among rocks, thus in 50 particles r_1 is valuable and in 50 particles r_2 is valuable. Hence, the belief $\bar{b}$ can be represented as $\{Pr(r_1 = 1) = Pr(r_2 = 1) = 50\%, p_c = (1,1), p_1 = (1,2), p_2 = (2,2)\}$. This can be lifted, using the feature map, as $F_\mathcal{F}(\bar{b}) = \{$`guess(1,50)`, `guess(2,50)`, `dist(1,1)`, `dist(2,2)`, `delta_x(1,0)`, `delta_x(2,1)`, `delta_y(1,1)`, `delta_y(2,1)`, `num_sampled(0)`$\}$.

Similarly, we can define an *action map* $F_\mathcal{A} : A \to \mathcal{H}_b(\mathcal{A})$. For instance, if the action $\bar{a}$ of the agent in Figure 2a is to check r_1, $F_\mathcal{A}(\bar{a}) = \{$`check(1)`$\}$. In this way, we can build the following CDPI corresponding to the specific situation depicted in Figure 2a:

$$\langle\langle F_\mathcal{A}(\bar{a}), \mathcal{H}_b(\bar{a}) \setminus F_\mathcal{A}(\bar{a})\rangle, F_\mathcal{F}(\bar{b})\rangle \tag{8}$$
$$=\langle\langle\{\texttt{check(1)}\}, \{\texttt{check(2)}\}\rangle, \{\texttt{guess(1,50)}, \texttt{guess(2,50)}, \texttt{dist(1,1)}, \texttt{dist(2,2)},$$
$$\texttt{delta\_x(1,0)}, \texttt{delta\_x(2,1)}, \texttt{delta\_y(1,1)}, \texttt{delta\_y(2,1)}, \texttt{num\_sampled(0)}\}\rangle$$

The included set e^{inc} ($F_\mathcal{A}(\bar{a})$) contains the observed action per each time step, while the excluded set e^{exc} ($\mathcal{H}_b(\bar{a}) \setminus F_\mathcal{A}(\bar{a})$) contains all other possible not observed realizations for that action. In this way, ILASP can learn non only *what to do* (given the current context), but also *which actions to avoid*. For the same reason, we also define CDPIs in the form:

$$\langle\langle\emptyset, \mathcal{H}_b(a)\rangle, F_\mathcal{F}(\bar{b})\rangle, \forall a \in A \setminus \{\bar{a}\} \tag{9}$$
$$=\langle\langle\emptyset, \{\texttt{sample(\_)}\}\rangle, \{\texttt{guess(1,50)}, \texttt{guess(2,50)}, \texttt{dist(1,1)}, \texttt{dist(2,2)}, \ldots\}\rangle$$
$$\cup\langle\langle\emptyset, \{\texttt{east}\}\rangle, \{\texttt{guess(1,50)}, \texttt{guess(2,50)}, \texttt{dist(1,1)}, \texttt{dist(2,2)}, \ldots\}\rangle$$
$$\ldots$$

for all not executed (i.e., not observed) actions.

4.3 Definition of the ILASP Task

We can now set up the ILASP problem of Definition 3, defining the search space S_M and the background knowledge B. B contains definitions of variables and constants, as reported in Section 3.3. The search space definition is crucial for the computational performance of ILASP (Law et al., 2018), since more candidate hypotheses need to be explored. In this paper, we are interested in policy specifications following the map $\Gamma : \mathcal{F} \rightarrow \mathcal{A}$, which lifts the classical policy map $\pi : \mathcal{B} \rightarrow A$ to the ASP representation formalism. Hence, in S_M we only consider normal axioms in the form:

$$\bigwedge_{i=1}^{n} f_i \rightarrow a, \forall a \in \mathcal{A} \tag{10}$$

being $f_i, \ldots, f_n \in \mathcal{F} \cup \{\mathtt{X} \lesseqgtr x\}$, where $\mathtt{X}$ is any variable with possible integer constant value x (e.g., `D, V` in rocksample, for distances and probabilities). In this way, ILASP can find lower and upper bounds to quantify the semantic information encoded in atoms as `dist(R,D)`, `guess(R,V)`, `delta_x(R,D)`, `delta_y(R,D)`, `num_sampled(V)`. To define the search space, we use the mode bias syntax provided by ILASP, which allows to specify the possible head and body atoms in rules of S_M. We then prune axioms where an integer variable $\mathtt{X}$ (`D`, `V`) appears, but $\mathtt{X} \lesseqgtr x$ is missing. In fact, a rule as `east :- dist(R,D)` is not meaningful until the possible values of `D` are specified by $\mathtt{D} \lesseqgtr d$. In this way, we also reduce the number of axioms in S_M, which improves the performance of conflict-driven constraint learning in ILASP. More details about the definition of the search space can be found in the experimental section and the public repository.

In Equation (10), we assume that the axioms for each action are *independent* of other actions (i.e., other actions cannot appear in the body of axioms). This is a reasonable and not restricting assumption, since all POMDP actions have an influence on the state representation (hence, the environmental features) through the transition map. For instance, in the scenario depicted in Figure 2a, where $Pr(r_1 = 1) = Pr(r_2 = 1) = 50\%$, the agent decides to check r_1 at time $t = 1$. Then, at $t = 2$, the particle distribution (hence the probability of rocks to be valuable) may be updated such that $Pr(r_1 = 1) = 80\%$, $Pr(r_2 = 1) = 50\%$. As a consequence, the agent may choose to move north to r_1 (Figure 2b), in order to sample it at $t = 3$ (Figure 2c). This results in the following CDPIs:

$$\langle\langle\{\mathtt{north}\}, \{\}\rangle, \{\mathtt{guess(1,80)}, \ \mathtt{guess(2,50)}, \ \mathtt{dist(1,1)}, \ \mathtt{dist(2,2)}, \tag{11}$$
$$\mathtt{delta\_x(1,0)}, \ \mathtt{delta\_x(2,1)}, \ \mathtt{delta\_y(1,1)}, \ \mathtt{delta\_y(2,1)}, \ \mathtt{num\_sampled(0)}\}\rangle$$
$$\langle\langle\emptyset, \{\mathtt{check(\_)}\}\rangle, \{\mathtt{guess(1,80)}, \ \mathtt{guess(2,50)}, \ \mathtt{dist(1,1)}, \ \mathtt{dist(2,2)}, \ \ldots\}\rangle$$
$$\ldots$$

namely, `north` action is executed in the context holding at $t = 2$; and:

$$\langle\langle\{\mathtt{sample(1)}\}, \{\mathtt{sample(2)}\}\rangle, \{\mathtt{guess(1,80)}, \ \mathtt{guess(2,50)}, \ \mathtt{dist(1,0)}, \ \mathtt{dist(2,1)},$$
$$\tag{12}$$
$$\mathtt{delta\_x(1,0)}, \ \mathtt{delta\_x(2,1)}, \ \mathtt{delta\_y(1,0)}, \ \mathtt{delta\_y(2,0)}, \ \mathtt{num\_sampled(0)}\}\rangle$$
$$\langle\langle\emptyset, \{\mathtt{check(\_)}\}\rangle, \{\mathtt{guess(1,80)}, \ \mathtt{guess(2,50)}, \ \mathtt{dist(1,1)}, \ \mathtt{dist(2,2)}, \ \ldots\}\rangle$$
$$\ldots$$

namely, `sample(1)` action is executed in the context holding at $t = 3$; These examples may be explained by the following ASP theory:

```
sample(R, t) :- north(t-1), delta_y(R, D, t-1), D ≥ 1.
north(t) :- check(R, t-1).
```

namely, rock `R` should be sampled after moving north, if rock `R` was at y-distance `D`≥ 1 from the agent at the previous time step `t-1`, and moving north should happen after checking `R`. The axioms above relate each action to the previous one, introducing an additional variable `t` representing the time step of execution. However, since `check(1)` at $t = 1$ modifies the probability of r_1 to be valuable from `guess(1,50)` to `guess(1,80)`, and `north` at $t = 2$ determines `dist(1,0)` (according to the transition map of the POMDP problem), the following equivalent axioms are derived:

```
sample(R) :- dist(R, D), D ≤ 0.
north :- guess(R, V), V ≥ 80, delta_y(R, D), D > 0.
```

This allows to consider single CDPIs separately, looking for policy specifications which match the context and the action *at the same time step*, rather than searching for temporal relations between actions. As a consequence, we are able to set up separate ILASP problems, one for each task action a, with S_M containing axioms in the form of Equation (10), but referred only to a. This significantly reduces the size of the search space and the computational performance of ILASP.

Finally, since POMDP traces contain only information about executed actions, there may occur time steps where other actions were possible, but less convenient and hence discarded. Then, we also want to learn weak constraints encoding preferences between multiple feasible actions. To this aim, assume that a normal axiom $\mathcal{R}$ is learned for action $a \in \mathcal{A}$, and $a^* \in \mathcal{H}_b(a)$ is the executed (ground) instance of a at a given time step under belief b (thus, it appears in e^{inc} of a CDPI in the form of Equation (8)). We generate CDPIs in the form:

$$\langle\langle\{\bar{a}\}, \emptyset\rangle, F_{\mathcal{F}}(\bar{b})\rangle \ @pr_{\bar{a}} \quad \forall \bar{a} \in \mathcal{H}_b(\mathcal{R} \cup F_{\mathcal{F}}(\bar{b}))$$

namely, the included set contains ground actions $\bar{a}$ which are entailed by $\mathcal{R}$, given the initial grounding $F_{\mathcal{F}}(\bar{b})$. Each CDPI comes with its preference value $pr_{\bar{a}}$ (see Section 3.4), which is set higher for a^* (which was actually executed). In other words, for each action, we generate CDPIs corresponding to alternative task realizations where another feasible action could have been executed, according to policy specifications discovered by ILASP. We then assign a preference weight to CDPIs, in order to penalize non observed realizations with respect to the actually executed action in the recorded trace.

For instance, in our simplified rocksample scenario, the rule $\mathcal{R}$ for $a =$`check(R)` is:

$$\texttt{check(R) :- guess(R, V), V} \leq \texttt{50.} \tag{13}$$

which would ground both `check(1)`, `check(2)` at $t = 1$ (Figure 2a). However, only the former is executed, hence $a^* = $ `check(1)`. We then build the following CDPIs

$$\langle\langle\{\texttt{check(1)}\}, \{\}\rangle, \{\texttt{guess(1,50)}, \texttt{guess(2,50)}, \texttt{dist(1,1)}, \ldots\rangle@pr_1$$
$$\cup\langle\langle\{\texttt{check(2)}\}, \{\}\rangle, \{\texttt{guess(1,50)}, \texttt{guess(2,50)}, \texttt{dist(1,1)}, \ldots\rangle@pr_2$$

with $pr_1 > pr_2$.

To learn preferences, we set up another ILASP task with ordered examples as in Law et al. (2016), with a search space (defined via mode bias) in the form:

$$:\sim \ \texttt{f}_1(\texttt{V}_1,\ldots,\texttt{V}_m),\ldots,\texttt{f}_n(\texttt{V}_1,\ldots,\texttt{V}_m),\texttt{a}(\texttt{V}_1,\ldots,\texttt{V}_m).[\texttt{w@pr, V}_1,\ldots,\texttt{V}_m] \ \forall a \in \mathcal{A} \quad (15)$$

where $f_i,\ldots,f_n \in \mathcal{F}$. $\texttt{V}_1,\ldots\texttt{V}_m$ are ASP variables, $\texttt{pr}$ is the priority level and $\texttt{w}$ is any integer ASP variable (e.g., $\texttt{D},\texttt{V}$). In our example scenario, this leads to learning the following weak constraint:

```
:∼ check(R), dist(R, D). [D@1, R, D]
```

meaning that the agent prefers sensing the closest rock (with the lowest weight $\texttt{D}$), if two rocks have the same probability to be valuable (≤ 50 from Equation (13)).

4.4 Logical Policy Specifications for Online POMDP Solving

ILASP generates policy specifications which can be used to drive the decision-making process of an online POMDP agent. In the following, we show how the set of ILASP axioms can be integrated as heuristics in POMCP and DESPOT. The generalization to their extensions (e.g., AdaOPS by Wu et al., 2021 and POMCPOW by Sunberg & Kochenderfer, 2018) is straightforward, since the underlying solving principle is analogous. With reference to our leading example in Figure 2, we obtain the following set of axioms for the observed actions:

$$0 \ \{\texttt{check(R)} : \texttt{guess(R, V)}, \ \texttt{V} \leq 50\} \ 2. \qquad\qquad\qquad (16)$$

```
:∼ check(R), dist(R, D). [D@1, R, D]
```

$$\texttt{sample(R)} \ \texttt{:- dist(R, D)}, \ \texttt{D} \leq 0. \qquad\qquad\qquad (17)$$

```
north :- guess(R, V), V ≥ 80, delta_y(R, D), D ≥ 1.
```

The first aggregate axiom states that multiple rocks may be sensed (at most 2 in our scenario), if they are valuable with probability $\leq 50\%$. However, the weak constraint suggests to check only the closest one (weight $\texttt{D}$ to be minimized). Other axioms specify that a rock should be sampled when the distance from it is null, and the agent should move north towards a rock $\texttt{R}$ which is at positive y-distance ($\texttt{D} \geq 1$) and is valuable with probability $\texttt{V} \geq 80\%$. Percentages of covered examples *cov*, i.e., CDPIs entailed by learned axioms, are equal to 66%, 100% and 100% for $\texttt{check(R)}$, $\texttt{sample(R)}$, $\texttt{north}$, respectively.

4.4.1 POLICY SPECIFICATIONS IN POMCP

As explained in Section 3.2.1, POMCP exploration of the belief-action space is performed with MCTS, specifically maximizing the value of the chosen action via UCT in Equation (1), or selecting the next action randomly in the rollout phase when a leaf node is encountered in the tree. We use learned policy heuristics in both phases of the exploration.

Specifically, at each root node in MCTS, we ground environmental features from the current belief according to map $F_{\mathcal{F}}$. For instance, in Figure 2a corresponding to belief $\bar{b} = \{Pr(r_1 = 1) = Pr(r_2 = 1) = 50\%, p_c = (1,1), p_1 = (1,2), p_2 = (2,2)\}$, the following atoms are grounded: $F_{\mathcal{F}}(\bar{b}) = \{\texttt{guess(1,50)}, \ \texttt{guess(2,50)}, \ \texttt{dist(1,1)}, \ \texttt{dist(2,2)}, \ \texttt{delta\_x(1,0)},$

delta_x(2,1), delta_y(1,1), delta_y(2,1), num_sampled(0)}. According to learned specifications in Equation (16), check(1) can be grounded. Only for ground actions, we introduce a prior in UCT, increasing the value $V(ha)$ to the same value of c, and $N(ha)$ to a fixed high value (this is domain-dependent, we empirically set it to 10 in our domains). This is the same as implying that we have already performed $N(ha)$ simulations in the current belief, and they achieved a good discounted return. When we reach a non-root node, the belief is assumed invariant with respect to the root node, hence environmental features corresponding to the unobservable state (guess(R,V) in rocksample) are kept the same, while the ones corresponding to the observable state (dist(R,D), delta_x(R,D), delta_y(R,D), num_sampled(V)) are updated according to the transition map during the simulation. This implies that the belief distribution is assumed invariant during the simulation process, which may affect the optimality of decision making in highly dynamic scenarios, where the environment may significantly change while the agent is still making its decision. However, as we will empirically show in Section 5, this does not significantly impact POMCP performance. In fact, after the optimal action is selected, the simulation process starts again at the subsequent time step, thus the belief is updated at each time step and consequently, so does the grounding of policy specifications.

When a leaf (unexplored) node is reached, the rollout step is invoked. Differently from Mazzi et al. (2023b), where we only adopted policy heuristics in UCT, we now exploit learned heuristics also in the rollout phase. Similarly to UCT, we first ground environmental features and generate ground (suggested) actions. We then define a discrete distribution over all actions, i.e., in the scenario of Figure 2, $\mathcal{A} = \{$east, north, south, west, check(1), check(2), sample$\}$. For each action, the weight in the probability distribution is set according to the outcome of learned policy specifications and the coverage percentages. Specifically, for each action $a_i \in \mathcal{A}$ we define the probabilistic weight as:

$$\rho_i = \begin{cases} cov_i & \text{if } a_i \in \mathcal{H}_b(\mathcal{R} \cup F_{\mathcal{F}}(\bar{b})) \\ \min\{cov_j\}_{ij1}^{|\mathcal{A}|} & \text{otherwise} \end{cases} \tag{18}$$

$\mathcal{R}$ being the set of learned specifications (Equation (16)). In other words, if an action a_i is entailed by ASP specifications $\mathcal{R}$ under the current belief interpretation $F_{\mathcal{F}}(\bar{b})$, its selection probability depends on the percentage of covered examples cov_i. Otherwise, the minimum cov_i is assigned to not entailed actions. Then, the probabilistic weight for each action is normalized as $\frac{\rho_i}{\sum_i \rho_i}$ and the rollout procedure chooses a random action according to these probabilities. In our example, in Figure 2a, check(1) is the only suggested action and has the minimal coverage percentage, thus $\rho_i \approx 12\% \; \forall a_i \in \mathcal{A}$. Hence, all actions have equal probability to be selected, similarly to the classical uninformed rollout. However, in Figure 2b, actions north and check(1) are grounded, hence selection probabilities are $\rho_{i \neq \text{north}} \approx 12\%$, $\rho_{\text{north}} \approx 16\%$, with a bias towards the action with the highest confidence level (i.e., north). In Figure 2c, sample(1) and check(2) are entailed at $t = 3$, thus $\rho_{i \neq \text{sample}} \approx 12\%$, $\rho_{\text{sample}} \approx 16\%$, with a bias toward the sampling action.

We remark that we use learned ASP axioms for *soft policy guidance*. In fact, at non-leaf nodes, branches corresponding to actions not entailed from ASP heuristics can still be explored by MCTS, since their probability is $\rho_i > 0$. This preserves the asymptotic optimality of POMCP (i.e., for infinite simulations). In Section 6, we will show empirically that this mitigates the impact of bad policy heuristics learned by ILASP.

4.4.2 Policy Specifications in DESPOT

As mentioned in Section 3.2.2, DESPOT performs anytime heuristics search in the sub-tree generated by K random scenarios approximating the belief-action tree. The fundamental ingredient of DESPOT is the definition of suitable lower ($l(b_0)$) and upper ($u(b_0)$) bounds on the value of the current root node b_0 in the tree. These bounds are used to drive the exploration process, hence they must be carefully handcrafted for each domain, encoding task knowledge in order to achieve good performance (Ye et al., 2017). In fact, the best action branches are chosen according to Equation (3), marginalizing over $u(b_0, a)$. Similarly, the exploration of the algorithm is driven by the minimization of $E(b')$, as expressed in (4). Since $\epsilon(b_0) = u(b_0) - l(b_0)$, choosing a good lower bound (i.e., close enough to the upper bound) can significantly drive DESPOT towards most promising branches in the belief tree, speeding up the exploration process until the termination condition $\epsilon(b_0) < \epsilon_0$ is met.

As explained by Ye et al. (2017), $u(b_0)$ can be computed with approximate *hindsight optimization* (Yoon et al., 2008), considering the *deterministic problem* with the initial node corresponding to the initial state $s_{\Phi_i,0}$ of each Φ_i of the K scenarios, then evaluating:

$$u(b_0) = \frac{1}{K} \sum_{i=1}^{K} u(s_{\Phi_i,0}) \tag{19}$$

where $u(s)$ is a *state-dependent* upper bound, often easier to define. For instance, in the pocman scenario:

$$u(s) = \sum_{f_i} \gamma^{1+d(f_i)} R(f_i) + \gamma^{1+\max_{f_i} d(f_i)} R_{end} \tag{20}$$

where $R(f_i)$ is the reward for eating food pellet f_i ($+1$), R_{end} is the reward for completing the level ($+100$) and $d(f_i)$ is the distance between the pocman agent and a food pellet. It is evident that the upper bound does not represent a belief-action map as the one we want to learn in this paper, but rather an *optimistic evaluation of the maximum achievable return from the current state*. Hence, the domain knowledge required for defining an upper bound is very basic, and it is related to the definition of the specific POMDP problem, e.g., the reward function. On the contrary, the lower bound is typically defined as the value achieved by a *default policy* π_0, thus we can exploit the learned heuristics for its definition. However, there is a fundamental difference between our default policy and the ones proposed for DESPOT (Ye et al., 2017). In fact, typically $l(b_0)$ is computed similarly to Equation (19) as:

$$l(b_0) = \frac{1}{K} \sum_{i=1}^{K} V_{\pi_0}(s_{\Phi_i,0})$$

In this way, it is possible to define a policy based only on *observable states* (specific belief realizations).

In contrast, ILASP learns policy specifications based on the *semantic interpretation* of the current belief (through the feature map $F_{\mathcal{F}}$). Specifically, at each belief node b_0, we compute the ground environmental atoms through map $F_{\mathcal{F}}$, and compute grounded actions

Table 1: Details of experiments. For each domain (RS = rocksample, Poc = pocman), we report main parameters which are either augmented (↑) or reduced (↓) with respect to the training setting, following the nomenclature in Sections 3.1.1-3.1.2. We also report main POMDP challenges analyzed in this paper: $|\pi|$ denoting the average planning horizon, and $|A|$ denoting the size of the action space.

EXP	Domain	Solver					
		POMCP	DESPOT		AdaOPS	POMCPOW	Deep RL
		C++	C++	Julia	Julia	Julia	Python
EXP-1 Training	RS($N = 12, M = 4$) $|A| = 10, |\pi| = 17$	✓					
	Poc($10 \times 10, G = 2, \rho_f = 0.5, \rho_g = 0.75$) $|A| = 4, |\pi| = 73$	✓					
EXP-2 Generalization	RS($N \uparrow, M \uparrow$) $|A| = 10 \dots 26, |\pi| = 17 \dots 67$	✓	✓	✓	✓	✓	
	Poc(size↑, $G \uparrow, \rho_g \uparrow, \rho_f \downarrow$) $|A| = 4, |\pi| = 73 \dots 85$	✓	✓				
EXP-3 Bad examples	RS($N \uparrow, M \uparrow$) $|A| = 10 \dots 14, |\pi| = 17 \dots 47$	✓	✓				
EXP-4 Few examples	RS($N \uparrow, M \uparrow$) $|A| = 10 \dots 14, |\pi| = 17 \dots 47$	✓	✓				
EXP-5 RL comparison	RS($N \uparrow, M \uparrow$) $|A| = 10 \dots 14, |\pi| = 17 \dots 47$	✓	✓				✓

according to policy specifications. Then, similarly to the rollout phase in POMCP explained in Section 4.4.1, we select a default action according to a discrete probability distribution weighted by the confidence levels ρ_i's of each action (derived from counterexamples found by ILASP). The only difference with respect to POMCP is that we set $\rho_i = 0$ in Equation (18) for actions *not entailed by policy heuristics*. In fact, the lower bound in DESPOT must suggest the minimum expected value of a given node. If this value is high, because it is generated from a good default action (suggested by the policy specifications), the gap $\epsilon(b_0) = u(b_0) - l(b_0)$ reduces, resulting in faster convergence to the termination condition $\epsilon(b_0) < \epsilon_0$ (see Section 3.2.2). Thus, our methodology aims at selecting *the best possible action* as the default one (hence computing the corresponding lower bound) at each DESPOT step. For instance, with reference to our leading example, in Figure 2a the policy specifications entails only check(1), which is selected as default action with $\rho_i = 100\%$. In Figure 2b north and check(2) are entailed, with selection probabilities $\rho_{\texttt{north}} \approx 60\%$, $\rho_{\texttt{check}} \approx 40\%$. Finally, in Figure 2c sample and check(2) are entailed, with selection probabilities $\rho_{\texttt{sample}} \approx 60\%$, $\rho_{\texttt{check}} \approx 40\%$.

5. Experiments

For both pocman and rocksample scenarios, we first use POMCP to generate 1000 POMDP traces to learn policy specifications from. Each trace is a sequence of belief-action pairs collected during the execution of one specific instance of the problem. All traces share a similar environmental setting, e.g., the same grid size and number of rocks for rocksample, and the same grid size and number of ghosts for pocman. POMCP is run *without any*

heuristics in this phase, so no bias is present in the generated traces. We empirically set 2^{15} online simulations and particles in POMCP[4], in order to generate good-quality traces with sufficiently high discounted return. Moreover, we only select traces achieving discounted return above the average over the total runs. This is a very simple and *domain-agnostic* criterion to select only the best traces and most relevant examples to learn good policy heuristics. The discount factor γ is set to 0.95 for all experiments.

The empirical analysis presented in this section investigates the following aspects:

- **EXP-1** the impact of learned policy heuristics on the performance of online POMDP solving, in scenarios *similar to the training setting* (i.e., with the same grid size and number of rocks for rocksample, and the same grid size and number of ghosts for pocman). This is a preliminary assessment of the quality of learned heuristics;

- **EXP-2** the generalization capabilities of learned policy specifications, e.g., in larger grids or with more rocks / ghosts;

- **EXP-3** the robustness of our methodology to low-quality heuristics, e.g., from bad examples or when the set of user-defined semantic features is incomplete;

- **EXP-4** the influence of the number of example traces on the convergence of ILASP learner and the performance of online POMDP solvers;

- **EXP-5** the comparison with black-box methods for POMDP planning and policy heuristics learning, e.g., neural network architectures.

For **EXP-1**, we consider POMCP solver. Similarly to Mazzi et al. (2023b), we conduct experiments with different numbers of particles *below* 2^{15}, but with the same number of rocks / ghosts and grid size. However, we introduce a novel ablation study, to highlight the role of policy heuristics in different phases of MCTS (namely, UCT and rollout). We also add the comparison with handcrafted policy specifications, as provided in former published research. For **EXP-2**, we consider POMCP, DESPOT and the more recent and powerful AdaOPS, which is meant to deal with large complex domains. We consider POMCPOW only as a baseline, because it has worse performance than the others in our experiments, as confirmed by Wu et al. (2021). We cannot perform a cross-planner validation, since the software implementations are different[5] and we decided to rely on publicly available packages to foster reproducibility. This does not limit the validity of our analysis, since the goal of **EXP-1** and **EXP-2** is to assess the performance of a given online POMDP solver *with and without the use of learned policy heuristics*, in order to validate the outcome of our proposed methodology. Moreover, for DESPOT and POMCP we analyze both rocksample and pocman. For AdaOPS, we only consider the rocksample scenario, since the pocman is not implemented in the available package. **EXP-3** and **EXP-4**, we consider both the rocksample and pocman scenarios, limiting our analysis to POMCP and DESPOT. In fact,

---

4. Following standard practice in POMDP, the numbers of particles and simulations coincide. Hence, from now on, we will refer to particles and simulations indistinguishably.

5. POMCP and DESPOT are implemented in C++, adapting the original code released by Silver and Veness (2010) and Ye et al. (2017), respectively. AdaOPS and POMCPOW are implemented in Julia, following the original codes at `https://github.com/LAMDA-POMDP/AdaOPS.jl.git` and `https://github.com/JuliaPOMDP/POMCPOW.jl`, respectively.

POMCPOW and AdaOPS are efficient extensions of POMCP and DESPOT, hence, results on the robustness of our methodology do not differ from the original algorithms. Finally, in **EXP-5** we evaluate the performance of Approximate Information State (AIS) learning in POMDPs, by Subramanian et al. (2022). We use the related code[6], restricting our analysis only to rocksample since pocman is not implemented. We summarize the settings and solvers for the 5 experiments in Table 1, highlighting in red the challenges of different scenarios related to the planning horizon $|\pi|$ and the size of the action space $|A|$.

In the following, in Section 5.1 we report the learned heuristics for the two domains, detailing the learning performance and highlighting the high interpretability, which allows to compare our specifications to the handcrafted ones provided by domain experts in Silver and Veness (2010), Ye et al. (2017). Then, we show quantitative experiments for **EXP-1** to **EXP-5** in the subsequent subsections. All tests in this paper are run on standard hardware equipped with 3.6 GHz e.g. Ryzen 7 and 32 GB RAM[7]. Moreover, we do not exploit any ASP reasoner during online planning with either DESPOT or POMCP. Instead, we implement learned heuristics in C++ syntax, to achieve the best online computational performance.

5.1 Learned Policy Specifications

We now report the learning performance and the policy specifications for rocksample and pocman.

5.1.1 Rocksample

In rocksample, the training scenario consists of a 12×12 ($N = 12$) grid with $M = 4$ rocks. Scenarios differ because of the random positioning and real values of rocks. The ASP encoding of the problem (i.e., the set of environmental features $\mathcal{F}$ and action atoms $\mathcal{A}$) is reported in Section 4.1. Additionally, we introduce the atom `target(R)`, meaning that rock `R` is the next to be sampled, which is considered both as an environmental feature and an action atom when defining ILASP search space as in Section 4.3. It allows to learn more compact rules (since ILASP does not support predicate invention, see Law, 2023), and can be easily derived from traces of execution, identifying sub-traces which do not contain `sample(R)` actions and assuming that all actions executed before sampling were directed to the final sampling. Moreover, we consider the `exit` action, which is equivalent to `east`, but has possibly different specifications.

Starting from 1000 traces of execution (generated within ≈ 12 min), we generate 8487 ILASP examples (corresponding to single steps of execution) for each possible head atom in $\mathcal{A}$ (after selecting only 492 traces with discounted return higher than the average). The

---

6. https://github.com/info-structures/ais
7. Since the hardware differs from the one used in our original work (Mazzi et al., 2023b), we have run also the experiments already reported there on the new machine, in order to perform a more fair empirical evaluation.

following ASP heuristics are found:

$$\text{east :- target(R), delta\_x(R,D), D} \geq 1. \tag{21}$$

```
west :- target(R), delta_x(R,D), D ≤ -1.
north :- target(R), delta_y(R,D), D ≥ 1.
south :- target(R), delta_y(R,D), D ≤ -1.
0 {target(R) : dist(R,D), not sampled(R), D ≤ 1.
    target(R) : guess(R,V), not sampled(R), 70 ≤ V ≤ 80} M.
:~ target(R), dist(R,D).[D@1, R, D]
:~ target(R), guess(R,V).[-V@2,R,V]
check(R) :- target(R), not sampled(R), guess(R,V), V ≤ 50.
check(R) :- guess(R,V), not sampled(R), dist(R,D), D ≤ 0, V ≤ 80.
sample(R) :- target(R), dist(R,D), D ≤ 0, not sampled(R), guess(R,V), V ≥ 90.
exit :- guess(R,V), V ≤ 40, not sampled(R), num_sampled(N), N ≥ 25.
exit :- dist(R,D), 5 ≤ D ≤ 8, not sampled(R), num_sampled(N), N ≥ 25.
```

The first 4 axioms define basic conditions for moving in the cardinal directions towards a chosen target rock. The aggregate rules for `target(R)` explain that an unsampled rock is chosen as a target when either it is close to the agent ($D \leq 1$) or probably valuable ($70 \leq V \leq 80$). Moreover, the weak constraints state that, among multiple possible targets, first the closest one should be chosen (priority 1 with cost `D`), then the most probably valuable (priority 2 with cost `-V`). Axioms for `check(R)` specify that it is worth sampling either when the chosen target rock has low probability of being valuable ($V \leq 50$), or the agent is at the location of an unsampled rock with value probability ≤ 80. In the first case, the heuristics pushes the agent to refine its belief for the most uncertain rocks, in order to compute the best strategy for the task. Similarly, in the second case, the effect is a *cautious* behavior of the agent, i.e., it wants to be almost sure about the value of the rock before picking it. When the agent is on the location of the target rock and the probability of the rock being valuable is ≥ 90, sampling should be executed. Finally, the agent should terminate the task, exiting the grid whenever the percentage of unsampled rocks is $\geq 25\%$ and either an unsampled rock has low value probability ($V \leq 40$) or it is far ($5 \leq D \leq 8$).

ILASP is able to find all axioms within $\approx 6\,\text{min}$, in a search space of ≈ 29500 axioms, setting a maximum length of 8 atoms for normal rules (Equation (10)) with mode bias. Specifically, we allow in the axioms 4 environmental features plus the comparison operators $V \lessgtr \bar{v}, D \lessgtr \bar{d}$. This is empirically set to balance between computational requirements (more features imply longer axioms in S_M, which influences the conflict-driven constraint learning approach of ILASP) and expressiveness of policy specifications. When learning weak constraints for `target(R)` atom, we set a maximum rule length (Equation (15)) of 6 since we do not need to include comparison operators. The percentage of not covered examples cov_i (used for action selection in the rollout phase of POMCP and for the definition of the default policy in DESPOT) is 65% for `north` and `south` actions, 57% for `east`, 73% for `west`, 84% for `exit`, 85% for `check(R)` and 65% `sample`.

The handcrafted heuristics proposed by Silver and Veness (2010) is defined as follows in the original paper:

> *Either check a rock whenever the value probability is uncertain ($< 100\%$) and it has been measured few times (< 5); or sample a rock if the agent is at its location and collected observations are more positive (good value) than bad; or move towards a rock with more good than bad observations.*

Thanks to the *interpretability of logical statements*, we can make a qualitative comparison between our learned policy specifications (21) and the handcrafted ones. Our rules state that a rock should be checked when the value probability is low ($\leq 50\%$), which is a possible consequence of having few measurements about that rock, as stated in the handcrafted heuristics. Moreover, our rules suggest to check a rock when the agent is at its location, even with high probability of the rock being valuable, in order to ensure a good pick. Our rules also indicate that a rock with a high probability of being valuable ($V \geq 90\%$) should be sampled when the agent is at its location, similarly to the handcrafted heuristics. Finally, ASP heuristics suggest to move towards candidate target rocks, i.e., the ones with high goodness probability ($V \geq 70\%$) as in the handcrafted heuristics, or even the closest ones to the agent. Hence, ILASP is able to find policy specifications which are similar to the handcrafted ones, but evaluate the belief distribution rather than single observations. Our methodology cannot capture the policy heuristics about the number of times a rock has been checked, since it would require more complex temporal expressiveness, e.g., the *until* operator from linear temporal logic. This will be discussed in Section 6.

5.1.2 POCMAN

In pocman, the training scenario consists of a 10×10 grid with $G = 2$ ghosts. The 1000 training instances generated differ on the basis of the motion of ghosts, which start always at the same location but wander randomly in the grid. Moreover, when close enough to the pocman agent (< 4 Manhattan distance), the ghosts can chase it with probability $\rho_g = 75\%$ probability. Each non-wall cell may contain a food pellet with probability $\rho_f = 50\%$ probability. We generate ILASP examples according to the following ASP environmental features $\mathcal{F}$:

- `food(C,D,V)`, representing the discrete probability (represented as a percentage) $V \in \{0, 10, \ldots, 100\}$ that a food pellet is within $D \in \mathbb{Z}$ Manhattan distance from the pocman in `C` cardinal direction, $C \in \{$`north, south, east, west`$\}$;

- `ghost(C,D,V)`, representing the discrete (percentage) probability $V \in \{0, 10, \ldots, 100\}$ that a ghost is within `D` Manhattan distance from the pocman in `C` cardinal direction;

- `wall(C)`, representing the presence of a wall in `C` cardinal direction.

The set $\mathcal{A}$ of ASP actions consists of the single atom `move(C)`, being $C \in \{$`north, south, east, west`$\}$.

Over 1000 traces of execution (generated within $\approx 6\,\mathrm{min}$), only the 460 with discounted return greater than the average are selected. Following the methodology presented in Section 4.2, we then have 43055 ILASP examples. The following policy heuristics is found:

$$\texttt{move(C) :- not wall(C), ghost(C, D, V), V} \leq \texttt{50, D} \leq \texttt{6.} \qquad (22)$$

meaning that the agent should move in direction C whenever there is no wall and the probability to find a ghost within Manhattan distance of 6 is $\leq 50\%$. ILASP is able to find axioms within ≈ 48 s, in a search space of ≈ 21000 axioms, generated considering a maximum normal rule length of 6 atoms (3 environmental features + comparison atoms) with mode bias. We do not learn weak constraints, hence we do not report details about the search space in Equation (15). The confidence level for move(C) action is $cov_i = 98\%$.

The handcrafted heuristics by Silver and Veness (2010) is:

> *Move in a direction where no ghost nor wall is seen, and avoid changing direction*
> *to minimize the number of steps.*

Similarly to the rocksample domain, we leverage on interpretability of ASP heuristics to make a qualitative comparison. The learned rule in Equation (22) captures the first part of the handcrafted heuristics, stating that the pocman should move in a direction where there is no wall and the probability of finding a ghost within 6 steps is low ($\leq 50\%$). The second handcrafted suggestion (avoid direction changes) is not represented in the environmental features for the task, since it requires memory of the past actions, hence introducing a temporal dimension in the learning problem. However, in the following we will see that this does not impact on the performance of the planner. Moreover, one fundamental difference is that the handcrafted heuristics depends on the observations of the agent (e.g., seeing a ghost), while the learned heuristics is based on the belief distribution about ghost locations, hence it more directly involves reasoning about the uncertain model of the world built through experience during task execution.

5.2 Results in the Training Setting (EXP-1)

We now evaluate the performance of POMCP with and without learned policy heuristics in scenarios similar to the training setting (i.e., same grid size and same number of rocks / ghosts), but with fewer online simulations and particles ($2^x \ \forall x \in \{8, 9, 10, 11, 12, 13, 14, 15\}$) to approximate the belief. This affects the accuracy of the policy found by POMCP, hence having a good heuristics is crucial to achieve high discounted return. For both rocksample and pocman, we perform an ablation study of our methodology, comparing 4 different algorithms:

- *POMCP*, i.e., standard POMCP without any policy heuristics(the one used also to generate learning traces);

- *POMCP + ASP MCTS*, i.e., standard POMCP with policy specifications added in the UCT phase, as explained in Section 4.4.1. This corresponds to the methodology proposed by Mazzi et al. (2023b);

- *POMCP + ASP full*, i.e., standard POMCP with policy specifications added in the UCT and rollout phases. This corresponds to the methodology proposed in this paper;

- *POMCP + preferred*, i.e., standard POMCP with handcrafted policy specifications (presented in Section 5.1) added in the UCT and rollout phases.
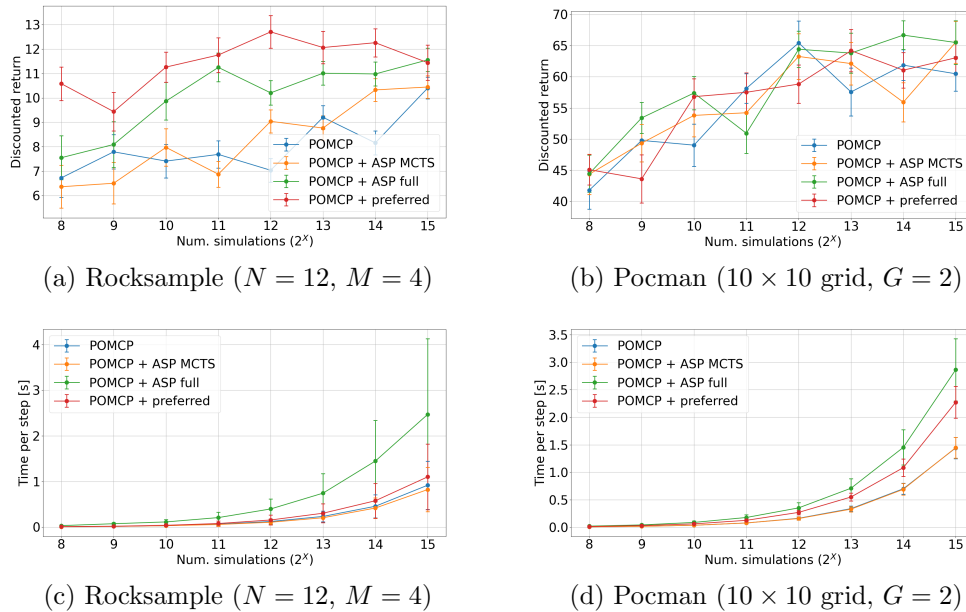
(a) Rocksample ($N = 12$, $M = 4$)

(b) Pocman (10×10 grid, $G = 2$)

(c) Rocksample ($N = 12$, $M = 4$)

(d) Pocman (10×10 grid, $G = 2$)

Figure 3: Discounted return (**top**) and computational time per step (**bottom**) with POMCP (mean ± standard deviation) for rocksample and pocman with different number of particles in the training setting (**EXP-1**).

5.2.1 Rocksample

The agent moves in a 12×12 ($N = 12$) grid with $M = 4$ rocks. For each number of particles, we run POMCP 50 times with random location and value of rocks. This experiment was partly already presented in our former paper (Mazzi et al., 2023b). However, for a more fair evaluation, here we set rocks sufficiently far one from the other, sub-dividing the original grid into M micro-grids, each one hosting at most one rock. In this way we avoid problem instances where two valuable rocks are almost adjacent, resulting in higher potential return for the agent.

Figure 3a shows the discounted return achieved by the agent. *POMCP + ASP full* (our methodology) is significantly better than *POMCP + ASP MCTS*[8] and *POMCP*, because it considers policy specifications *also in the rollout phase*. The performance are slightly worse than *POMCP + preferred*. This is probably due to the different knowledge encoded in the handcrafted heuristics. For instance, from Section 5.1, *POMCP + preferred* is driven to perform check(R) *if R was checked less than 5 times*. This temporal information requires

---

8. Notice that these results differ from Mazzi et al. (2023b) because that paper focued only on the impact of policy heuristics *in UCT phase*, hence the rollout (both for *POMCP* and *POMCP + ASP MCTS*) was driven by the handcrafted heuristics proposed by Silver and Veness (2010). In this paper, we introduce learned heuristics in the rollout phase, hence the standard *POMCP* is completely uninformed (i.e., it does not use any heuristics, hence implementing standard UCT with no prior initialization of $N(ha)$ and uniformly random rollout). Similarly, *POMCP + ASP MCTS* performs random rollout. Moreover, the experiments in our former paper assumed a different location of rocks, which were not equally distributed. This affected the average discounted return, since two valuable rocks might be adjacent on the map, thus potentially helping the agent.

to track the history of the task and cannot be captured by simply looking at traces of execution. On the other hand, while checking a rock is costless for the agent, multiple checks affect the plan length, hence the discounted return.

Figure 3c shows the computational time per step taken by the agent to choose the next action. Using policy heuristics also in the rollout, *POMCP + ASP full* takes more time as the number of simulations increases. However, comparing the performance of *POMCP + ASP full* and *POMCP* in Figure 3a, our algorithm requires only 2^{10} simulations to achieve the same performance as *POMCP* with 2^{15} simulations. Thus, from Figure 3c, the computational time per step is significantly reduced even with respect to pure POMCP (*POMCP + ASP full* with 2^{10} simulations requires ≈ 0.1 s per step, while *POMCP* with 2^{15} simulations requires ≈ 1 s per step). *POMCP + preferred* does not take much time because the policy heuristics *only considers observations*, while our heuristics takes into account the full belief distribution.

5.2.2 POCMAN

The agent moves in a 10×10 grid with $G = 2$ rocks. Each non-wall cell may have a food pellet with 50% probability, while ghosts can chase the agent with 75% probability. For each number of particles, we run POMCP 50 times.

Figures 3b-3d show that there is no significant difference between the different algorithms, especially in terms of discounted return. In fact, given the lower number of actions (only 4) with respect to rocksample (9 with $M = 4$), the influence of the heuristics in POMCP is minor. Finally, as stated by Silver and Veness (2010), Ye et al. (2017), the main challenge of pocman lies in the long planning horizon (for comparison, the rocksample agent on a 12×12 grid with 4 rocks executes ≈ 17 steps per execution, while the pocman agent in the small domain takes ≈ 73 actions), thus varying the number of simulations and particles to approximate the belief is not crucial for POMCP performance (over a small action space).

5.3 Generalization to Unseen Scenarios (EXP-2) - Rocksample

We now assess the performance of learned policy specifications in scenarios *different from the training setting*. Specifically, in the rocksample domain we increase the grid size N (affecting the planning horizon) and the number of rocks M (affecting the size of the action space). This aims at showing the generality of our heuristics, especially in more complex environments involving larger actions spaces and planning horizons. We extend our analysis to different online POMDP solvers, including POMCP, DESPOT and their more recent extensions POMCPOW (Sunberg & Kochenderfer, 2018) and AdaOPS (Wu et al., 2021), dealing with larger action spaces and state/observation spaces, respectively.

5.3.1 POMCP

As a first experiment in POMCP, we increase the number of rocks ($M = 8$) on a 12×12 grid, and evaluate performance as the number of online simulations decreases. Figures 4a, 4d show similar results as Figures 3a, 3c (**EXP-1**), i.e., *POMCP + ASP full* achieves better discounted return than *POMCP* and *POMCP + ASP MCTS*, though slightly worse than *POMCP + preferred*. Similarly our methodology requires the highest computational time
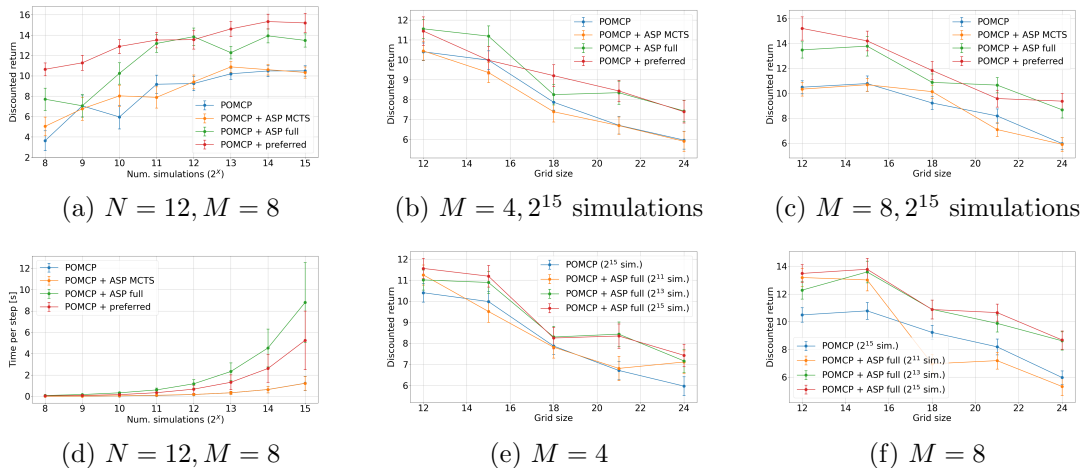
(a) $N = 12, M = 8$ (b) $M = 4, 2^{15}$ simulations   (c) $M = 8, 2^{15}$ simulations

(d) $N = 12, M = 8$ (e) $M = 4$ (f) $M = 8$

Figure 4: POMCP performance (mean ± standard deviation) for rocksample with different number of rocks and grid size (**EXP-2**).

per step (given the same number of simulations), but the overall computational effort is lower to achieve the same performance as *POMCP* with 2^{15} particles, requiring only 2^{10} particles (hence, *POMCP+ASP full* requires ≈0.2 s, while *POMCP* ≈1.1 s).

Then, in Figures 4b, 4c we evaluate the contribution of learned policy specifications in larger grid sizes N (x-axis) and with more rocks ($M = 4$ and $M = 8$, respectively), assuming a fixed number of particles and online simulations equal to 2^{15}. In both cases, *POMCP + ASP full* achieves comparable performance to *POMCP + preferred*, significantly outperforming *POMCP* and *POMCP + ASP MCTS*, especially at the largest grid size ($N = 24$).

In Figures 4e, 4f ($M = 4$ and $M = 8$, respectively), we show that this trend is confirmed even when the number of online simulations is lowered. In particular, *POMCP + ASP full* achieves similar discounted return even with 2^{13} particles, while its performance becomes comparable (Figure 4e) or slightly lower (Figure 4f) than *POMCP* when only 2^{11} simulations are used. Then, these experiments show that the use of learned heuristics in the rollout phase reduces the number of particles and online simulations required by POMCP, improving the planning performance even in unseen scenarios with longer planning horizon (on average 47 steps per execution) and larger action space (14 actions with 8 rocks).

We finally run a test in a very challenging rocksample domain, with $N = M = 20$. Here, *POMCP* achieves discounted return 9.83 ± 0.75, while *POMCP + ASP full* reaches 13.99 ± 0.81. *POMCP + preferred* achieves 16.32 ± 0.71, but at a higher computational time per step, because the observation space becomes significantly large. In fact, it takes 34.29 ± 7.50 s, while *POMCP + ASP full* requires only 15.84 ± 3.55 s. This shows that learned policy specifications are slightly less accurate and informative than handcrafted ones, as explained in former sections. However, in very large maps with very long planning horizons and many actions (26 actions available and 67 average steps executed with $M = 20$ rocks), they reach a suitable tradeoff between computational effort and performance in POMCP.
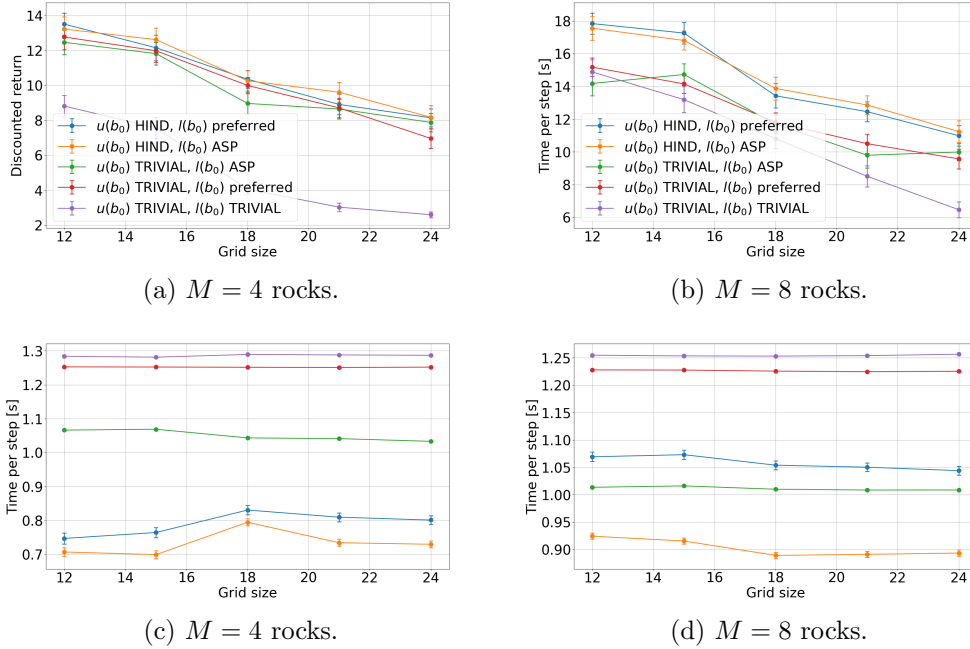
(a) $M = 4$ rocks.

(b) $M = 8$ rocks.

(c) $M = 4$ rocks.

(d) $M = 8$ rocks.

Figure 5: Discounted return (**top**) and computational time per step (**bottom**) with DESPOT **C++** (mean $\pm$ standard deviation) for rocksample in larger grids and with more rocks (**EXP-2**).

5.3.2 DESPOT

Following Ye et al. (2017), in DESPOT we consider two different upper bounds $u(b_0)$:

- *HIND*, corresponding to the hindsight approximation proposed by (Yoon et al., 2008, see Section 4.4.2);

- *TRIVIAL*, corresponding to the uninformed bound $u(b_0) = \frac{R_{max}}{1-\gamma}$, with R_{max} as the maximum task reward ($+10$ in rocksample). This is an important case study, because hindsight approximation may be computationally prohibitive to compute in large scenarios (see Section 4.4.2).

We then evaluate 3 different lower bounds $l(b_0)$:

- *ASP*, where the default policy is selected according to the learned policy specifications;

- *TRIVIAL*, where a default action is always chosen (moving east for rocksample);

- *preferred*, where the default policy is selected according to the handcrafted heuristics used for POMCP.

We then consider different combinations of lower and upper bounds in our experiments (from now on, we will use the syntax $l(b_0) + u(b_0)$ to refer to different planning combinations). We keep the number of scenarios fixed to $K = 500$ (the standard value used by Ye et al.,

2017), since in our experiments changing this parameter did not significantly impact the performance.

In Figures 5a,5c, we show the performance with 4 rocks in larger grids, while results with 8 rocks are reported in Figures 5b,5d. In general, DESPOT with *HIND* upper bound achieves the highest performance, both with *preferred* and *ASP* lower bounds. Thus, when the upper bound is accurate, the solver is able to compute a good-quality plan. However, when the upper bound is uninformative (*TRIVIAL*), the policy specifications to determine the lower bound become crucial. *ASP* lower bound is able to achieve the same performance as the handcrafted *preferred* one. On the contrary, DESPOT with *TRIVIAL* lower bound is significantly worse, especially as the grid size increases to $N = 24$. It is interesting to note that *ASP* lower bound minimizes the computational time per step. In particular, from Figure 5c (similar results can be observed from Figure 5d) the combination *ASP + TRIVIAL* (green curve) requires ≈ 1.1 s per step for all grid sizes, while *preferred + TRIVIAL* requires ≈ 1.30 s (similarly to *TRIVIAL + TRIVIAL*, i.e., the worst combination). Even when the optimal upper bound *HIND* is used, *ASP + HIND* requires less time than *preferred + HIND*, especially with 8 rocks in Figure 5d (≈ 0.9 s vs. ≈ 1.05 s).

We finally evaluate DESPOT in the challenging rocksample scenario with $N = M = 20$. Here, we could not compute the hindsight upper bound, since it caused memory overload on our machine, given the very large domain size. Hence, only the *TRIVIAL* upper bound is considered. Similarly to POMCP, *ASP* lower bound achieves slightly worse discounted return than *preferred* ($\approx 10.18 \pm 0.89$ vs. $\approx 12.81 \pm 0.57$). However, learned heuristics require significantly less computational time per step ($\approx 1.43 \pm 0.07$ s vs. $\approx 2.22 \pm 0.25$ s).

5.3.3 AdaOPS and POMCPOW

With AdaOPS and POMCPOW, we report similar experiments to the previous section, varying the number of rocks M and the grid size N. In particular, we implement ASP heuristics into the original code by Wu et al. (2021), considering only *HIND* and *TRIVIAL* upper bounds, *ASP* and *TRIVIAL* lower bounds (since handcrafted lower bounds are not implemented in Julia). We then report POMCPOW[9] performance (in terms of discounted return) only as a baseline, since it performs significantly worse than AdaOPS.

Figure 6 shows the performance of AdaOPS. The results confirm the ones evidenced in C++ with DESPOT. The use of learned policy specifications for the default action and the lower bound (*ASP*) does not significantly impact the discounted return achieved with the optimal *HIND* upper bound, both with 4 and 8 rocks (Figures 6a-6b, respectively). However, *ASP* lower bound improves the performance when the uninformed upper bound (*TRIVIAL)* is used. In terms of computational time per step (Figure 6c for 4 rocks, Figure 6d for 8 rocks), the *ASP* lower bound improves the efficiency of the planning process with the good upper bound *HIND*. With the uninformed upper bound *TRIVIAL*, the solver requires the same computational effort with any lower bound, hence the curve for *TRIVIAL + TRIVIAL* is superimposed to *ASP + TRIVIAL* in Figures 6c-6d. In the 20×20 grid with 20 rocks, when *HIND* upper bound[10] is used *ASP* does not bring any benefit to

---

9. We do not perform action widening in POMCPOW, since the action space is discrete and this resulted in better performance in our experiments.
10. In Julia, the hindsight approximation can be computed on the available hardware, differently from C++ code, probably due to the different software implementation.
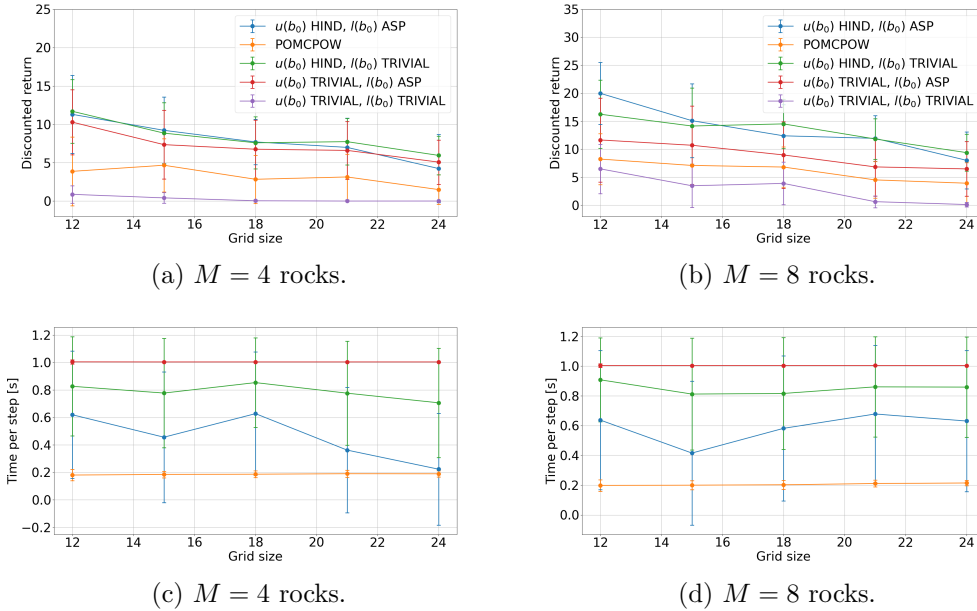
(a) $M = 4$ rocks.

(b) $M = 8$ rocks.

(c) $M = 4$ rocks.

(d) $M = 8$ rocks.

Figure 6: Discounted return (**top**) and computational time per step (**bottom**) with AdaOPS and POMCPOW (mean ± standard deviation) for rocksample in larger grids and with more rocks (**EXP-2**).

AdaOPS. However, when the uninformed upper bound *TRIVIAL* is adopted, *ASP* lower bound achieves significantly higher discounted return than *TRIVIAL* (10.81±8.30 vs. 0.03± 0.54).

We also perform further experiments with Julia version of DESPOT, confirming the C++ results. Hence, they are reported only in the public repository for brevity.

5.4 Generalization to Unseen Scenarios (EXP-2) - Pocman

In the pocman domain we assess generalization by increasing the grid size to 17×19, as proposed by Silver and Veness (2010), Ye et al. (2017). This further increases the planning horizon with respect to the 10×10 grid size used to generate training traces in Sections 5.1-5.2.2, rising from 73 to 85 steps per execution, on average. As a comparison, in the rocksample domain on a 20×20 grid with 20 rocks, the agent takes on average 67 steps.

We also evaluate the performance when some parameters of the task are changed, i.e., the aggressivity and number of ghosts and the percentage of food pellets. This may significantly change the behavior of the agent with respect to the training setting, thus the quality and generality of policy specifications is fundamental.

We only report results in C++ with POMCP and DESPOT, since this domain is not implemented in Julia.

(a)

(b)
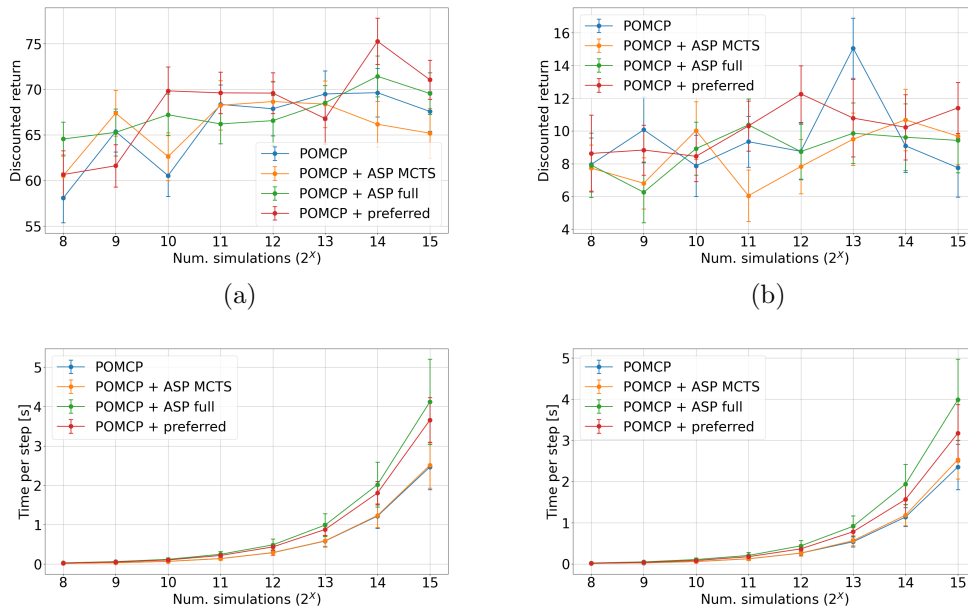
(c) 17×19 grid, $G = 4, \rho_g = 75\%, \rho_f = 50\%$. (d) 17×19 grid, $G = 4, \rho_g = 100\%, \rho_f = 20\%$.

Figure 7: Discounted return (**top**) and computational time per step (**bottom**) with POMCP (mean ± standard deviation) for pocman in more challenging domains (**EXP-2**).

5.4.1 POMCP

In Figures 7a, 7c, we show POMCP performance in the larger map (17×19) with $G = 4$ ghosts, varying the number of online simulations and particles. In Figures 7b-7d, we further complicate the setting for the agent, increasing the probability that ghosts chase the agent up to $\rho_g = 100\%$ and decreasing the probability to find food pellets to $\rho_f = 20\%$. In both cases, there is no significant difference between the performance of different POMCP versions, i.e., with or without learned and handcrafted heuristics. This confirms the results observed in Section 5.2.2 in the training setting, proving that even good policy heuristics have small impact on POMCP performance in a scenario with very long planning horizon. It is still interesting to note, however, that learned heuristics do not deteriorate the performance of the planner.

5.4.2 DESPOT

Table 2 shows the performance of DESPOT in different maps and with standard values for ghost aggressivity ($\rho_g = 75\%$) and food probability ($\rho_f = 50\%$), considering the different upper and lower bound combinations explained for rocksample. In general, using ASP as a lower bound leads to similar performance as the best configuration ($preferred + HIND$) even when the $TRIVIAL$ upper bound is used, both in terms of computational time per step and discounted return. It is interesting to note the significant improvement with respect to

Table 2: **EXP-2**: Pocman results with DESPOT (mean $\pm$ standard deviation) in nominal conditions ($\rho_f = 50\%$, $\rho_g = 75\%$).

$l(b_0)$	$u(b_0)$	$10 \times 10, G = 2$		$17 \times 19, G = 4$	
		Disc. ret.	**Time/step [s]**	**Disc. ret.**	**Time/step [s]**
ASP	HIND	55.64 ± 2.41	1.09 ± 0.001	70.00 ± 2.82	1.12 ± 0.002
Preferred	HIND	52.37 ± 3.52	1.10 ± 0.002	72.56 ± 2.04	1.12 ± 0.002
ASP	TRIVIAL	53.42 ± 3.32	1.09 ± 0.002	64.91 ± 2.20	1.11 ± 0.002
Preferred	TRIVIAL	51.52 ± 3.30	1.11 ± 0.002	71.31 ± 2.03	1.12 ± 0.002
TRIVIAL	TRIVIAL	2.67 ± 3.16	1.42 ± 0.003	-8.63 ± 5.04	1.35 ± 0.004

Table 3: **EXP-2**: Pocman results with DESPOT (mean $\pm$ standard deviation) in challenging conditions ($\rho_f = 20\%$, $\rho_g = 100\%$).

$l(b_0)$	$u(b_0)$	$10 \times 10, G = 2$		$17 \times 19, G = 4$	
		Disc. ret.	**Time/step [s]**	**Disc. ret.**	**Time/step [s]**
ASP	HIND	17.56 ± 3.53	1.07 ± 0.001	10.63 ± 1.68	1.09 ± 0.002
Preferred	HIND	12.98 ± 3.07	1.08 ± 0.002	16.30 ± 1.58	1.12 ± 0.002
ASP	TRIVIAL	11.15 ± 3.17	1.07 ± 0.001	13.28 ± 1.63	1.08 ± 0.002
Preferred	TRIVIAL	14.88 ± 3.90	1.08 ± 0.002	14.67 ± 1.66	1.11 ± 0.002
TRIVIAL	TRIVIAL	9.29 ± 4.33	1.33 ± 0.002	-4.84 ± 2.82	1.35 ± 0.004

TRIVIAL + TRIVIAL in both maps, thus showcasing the high quality and the fundamental role of learned heuristics for planning performance.

Results in Table 3 partly confirm these observations when the parameters of the task become more challenging ($\rho_f = 20\%$, $\rho_g = 100\%$). In fact, in the smaller map[11] the *ASP* lower bound performs better than *preferred* when *HIND* upper bound is used (17.56 ± 3.53 vs. 12.98 ± 3.07), but is worse with the *TRIVIAL* upper bound (11.15 ± 3.17 vs. 14.88 ± 3.90). Conversely, in the larger map *ASP* lower bound achieves similar performance as *preferred* with the *TRIVIAL* upper bound (13.28 ± 1.63 vs. 14.67 ± 1.66), but worse performance in combination with *HIND* upper bound (10.63 ± 1.68 vs. 16.30 ± 1.58). Overall, learned *ASP* heuristics always achieve significantly superior discounted return than *TRIVIAL + TRIVIAL*, thus showing the quality of learned policy specifications.

5.5 Influence of Quality and Quantity of Example Traces on Planning Performance (EXP-3-EXP-4)

In Section 5.1, we have assumed that 1000 traces of execution are generated from a *high-quality* planner, i.e., POMCP with 2^{15} particles and online simulations. In very complex scenarios, this may be infeasible due to the required time to generate traces (in POMCP,

---

11. We remark that even though the map size is the same as the scenario for generating learning traces, the task is more challenging, because of the fewer food pellets and the highest probability to get caught by ghosts (as evidenced by the significant difference of discounted returns between Tables 2-3).
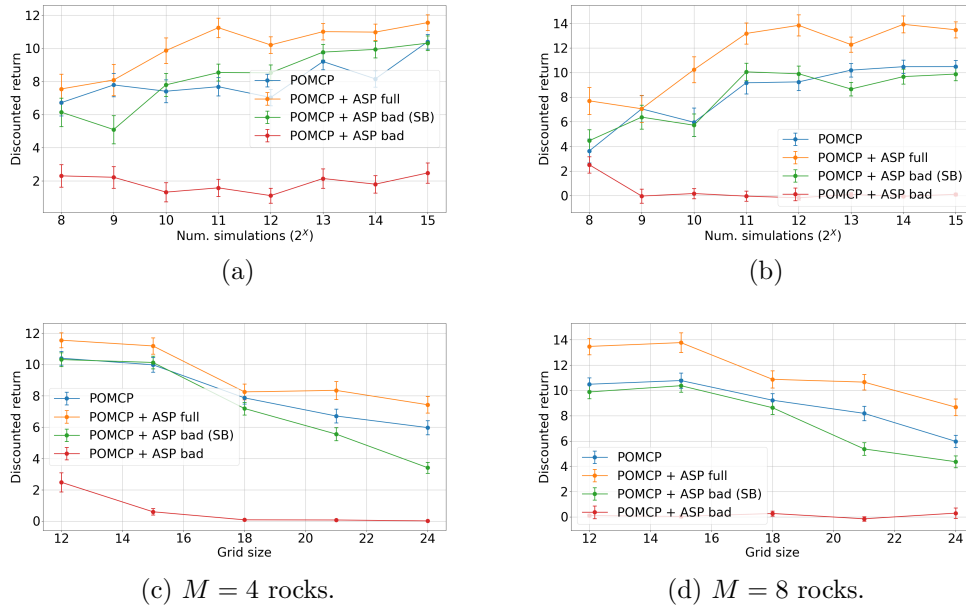
Figure 8: Results (mean $\pm$ standard deviation of the discounted return) for rocksample in POMCP with policy heuristics learned from bad examples (**EXP-3**). **Top**: 12×12 grid with different number of simulations. **Bottom**: 2^{15} simulations with increasing grid size.

the number of particles affects the computational time per step; moreover, larger scenarios require longer planning horizons, hence, more steps of computation from all solvers). In addition, the agent may not be able to acquire useful and sufficient experience in a limited number of scenarios, hence some actions and task configuration may be underrepresented in the training traces (e.g., with very large action or state spaces).

In this section, we then investigate the impact of the quality and quantity of training example traces on the quality of learned policy heuristics. Specifically, we study the effect on the planning performance when: i) low-quality example traces are generated (**EXP-3**), and ii) fewer example traces are available (**EXP-4**).

5.5.1 EFFECT OF LOW-QUALITY EXAMPLES (**EXP-3**)

We generate 1000 training traces with 2^{10} particles in POMCP. From Figures 3a, 3b, this is where the discounted return achieved by pure POMCP (blue curves) drops significantly in both rocksample and pocman. In this way, we learn the following ASP heuristics for

rocksample:

```
east :- target(R), delta_x(R,D), D ≥ 1.                              (23)
west :- target(R), delta_x(R,D), D ≤ -1.
north :- target(R), delta_y(R,D), D ≥ 1.
south :- target(R), delta_y(R,D), D ≤ -1.
target(R) :- dist(R,D), not sampled(R), D ≤ 2.
check(R) :- guess(R,V), not sampled(R), dist(R,D), D ≤ 0, V ≤ 40.
sample(R) :- target(R), dist(R,D), D ≤ 0, not sampled(R), guess(R,V), V ≥ 70.
exit :- guess(R,V), V ≤ 40, num_sampled(N), N ≥ 25.
```

With respect to good heuristics in Equation (21), axioms for `exit` and `target(R)` depend on a smaller subset of the semantic features. In fact, `exit` only depends on the number of sampled rocks, hence the agent is pushed towards terminating the task whenever 1/4 rocks have been sampled. A rock is chosen as a target only depending on its distance, ignoring its expected value. This affects the validity of axioms for actions of motion which, in turn, depend on `target(R)`. As a consequence, the tests here reported not only analyze the influence of the quality of examples on the planning performance, but also the impact of *bad definition of environmental features $\mathcal{F}$*. In fact, though they can be easily retrieved from the POMDP definition (transition and reward maps), in very complex domains some high-level domain concepts may be still missed.

In the pocman domain, the learned heuristics are instead the same, hence we report results with the bad heuristics only for rocksample. In POMCP, we want to verify that our soft policy guidance methodology (Section 4.4.1) preserves the performance of POMCP when bad heuristics are learned, thanks to the non-null weights for selecting actions in Equation (18). For this reason, we consider 4 different solver configurations:

- *POMCP* and *POMCP + ASP full*, with good policy heuristics from Equation (21), as for **EXP-1**-**EXP-2**;

- *POMCP + ASP bad (SB)*, where bad policy specifications in Equation (23) are implemented both in rollout and UCT, following the soft policy guidance approach proposed in Section 4.4.1; *POMCP + ASP bad*, where the methodology in Section 4.4.1 is modified such that the probability of selecting rollout actions not entailed by bad policy specifications is set to 0. In other words, the *soft policy guidance approach* is omitted from our methodology.

Results in Figure 8 highlight the importance of the soft policy guidance approach. In fact, *POMCP + ASP bad (SB)* performs nearly as *POMCP* with any number of rocks, simulations and grid sizes (a slightly more pronounced decrease is observed in Figures 8c, 8d, where the planning horizon increases with the grid size). On the contrary, *POMCP + ASP bad* performs dramatically worse than other solvers, since rollout actions are pruned away according to bad policy specifications. Clearly, *POMCP + ASP full* (with good heuristics) performs better.

We perform a similar analysis in DESPOT (C++): we consider both the *TRIVIAL* and *HIND* upper bounds, and evaluate the difference in performance when good and bad
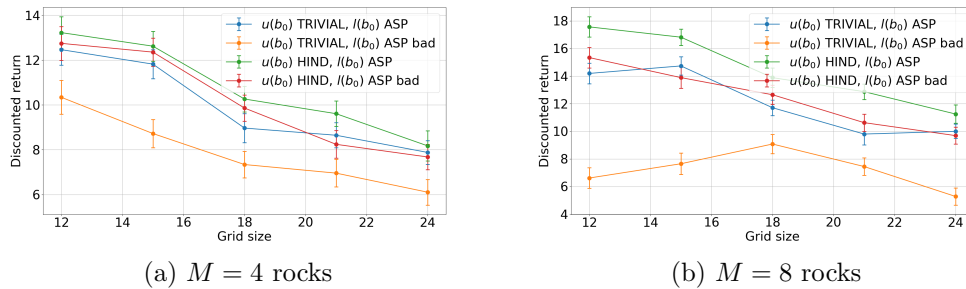
(a) $M = 4$ rocks

(b) $M = 8$ rocks

Figure 9: Discounted return (mean ± standard deviation) with DESPOT (C++) for rocksample on grids with different sizes, with policy heuristics learned from bad examples (**EXP-3**).

Table 4: **EXP-4**: Discrepancy between policy specifications for rocksample domain, learned from different numbers of traces / examples (with respect to final specifications in Equation (21)). Results for `target(R)` do not consider the weak constraints for simplicity.

Action	Axiom distance (%)			
Fraction of 1000 traces	20%	40%	60%	80%
north	25	50	0	0
south	83	80	0	0
east	25	0	0	0
west	0	0	0	0
target(R)	38	75	57	0
check(R)	78	0	0	0
exit	82	67	82	0
sample(R)	0	0	0	0
Average	41	34	17	0

ASP heuristics are adopted in the lower bound (*ASP* and *ASP bad*, respectively). Results reported in Figure 9 show that when the informed *HIND* upper bound is used, the effect of bad policy heuristics is neglectable. However, the combination *ASP bad + TRIVIAL* performs significantly worse than others. In fact, DESPOT strongly depends on the quality of heuristics in the bounds, hence it is not inherently robust to bad or incomplete specifications.

5.5.2 EFFECT OF FEW EXAMPLES (**EXP-4**)

We now vary the number of training example traces, selecting subsets of decreasing size from the original 1000 traces generated with 2^{15} particles in POMCP. Specifically, we consider random subsets containing $\{20, 40, 60, 80\}\%$ of the total 1000 traces. This corresponds to $\approx \{1700, 3400, 5100, 6800\}$ examples in rocksample, and $\approx \{9000, 18000, 27000, 36000\}$ for pocman, where the original number of examples (CDPIs) was ≈ 8500 and ≈ 45000, respectively. As for **EXP-3**, the learned policy specifications do not change in pocman,
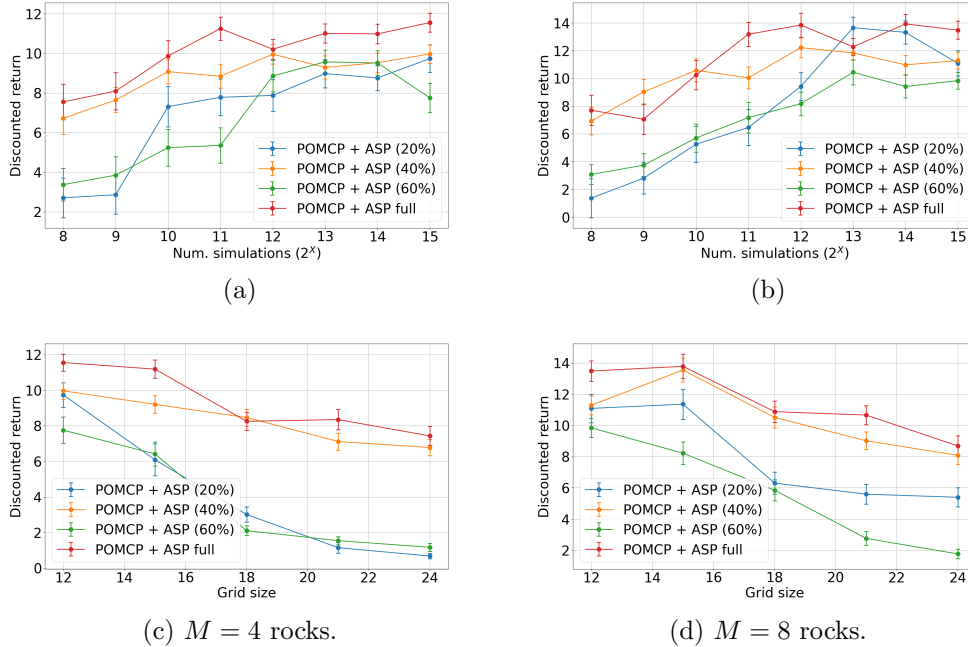
(a)

(b)

(c) $M = 4$ rocks.

(d) $M = 8$ rocks.

Figure 10: Results (mean $\pm$ standard deviation of the discounted return) for rocksample in POMCP with policy heuristics learned from few examples (**EXP-4**). **Top**: 12×12 grid with different number of simulations. **Bottom**: 2^{15} simulations with increasing grid size.

hence we only report results for rocksample. For brevity, we report learned heuristics for all training sizes only in Appendix A.

First, we investigate the syntactic discrepancy between policy specifications learned from different example sets. To this aim, we evaluate the *distance between axioms* generated from a percentage $X \in \{20, 40, 60, 80\}\%$ of the original 1000 training traces, with respect to the axioms generated from the total traces. Specifically, given 2 specifications $\mathcal{R}_1, \mathcal{R}_2$, corresponding to a same action atom $\mathsf{a} \in \mathcal{A}$ and involving independent sets of environmental feature atoms, $\mathcal{F}_1, \mathcal{F}_2$, we define the distance between the specifications as:

$$d(\mathcal{R}_1, \mathcal{R}_2) = \frac{|\mathcal{F}_1 \cup \mathcal{F}_2| - |\mathcal{F}_1 \cap \mathcal{F}_2|}{|\mathcal{F}_1 \cup \mathcal{F}_2|}$$

denoting with $|\cdot|$ the cardinality of a set. For instance, assume that $\mathcal{R}_1$ is the specification for action `south` in Appendix A.1, and $\mathcal{R}_2$ is the specification for `south` reported in Equation (21). Then, $D(\mathcal{R}_1, \mathcal{R}_2) = 0.5$, since $\mathcal{F}_1 = \{\texttt{delta\_y(R, D)}, \texttt{target(R)}, \texttt{D==-2}\}$, $\mathcal{F}_2 = \{\texttt{target(R)}, \texttt{delta\_y(R, D)}, \texttt{D} \leq \texttt{-1}\}$. Hence, $|\mathcal{F}_1 \cup \mathcal{F}_2| = 4, |\mathcal{F}_1 \cap \mathcal{F}_2| = 2$. Table 4 shows that, on average, the distance with respect to policy heuristics learned from the full dataset of traces (Equation (21)) decreases as the amount of examples increases, and it becomes null when 80% of the dataset is considered. This empirically demonstrates the convergence of ILASP to a set of stable policy specifications, as more examples are available.

Figure 10 shows results in POMCP, for different number of rocks (M) and grid sizes (N). For any number of rocks, the performance of policy heuristics learned from different

761

percentages of traces converges to the best heuristics learned from 100% of the dataset *POMCP + ASP full*, as the number of online simulations increases (Figures 10a-10b). This again confirms that the asymptotic optimality of POMCP is guaranteed by the soft policy guidance approach, as the number of online simulations and particles for belief approximation increases. Policy specifications learned from 20% and 60% of the initial traces perform significantly worse than the ones learned from 40% of the traces. This is even more evident as the grid size increases (Figures 10c, 10d). Looking at learned heuristics reported in Appendix A, we can exploit *interpretability of logic specifications* to easily identify a possible reason. In fact, both for specifications learned from 20% and 60% of traces, the agent is incentivized to choose a rock as a target (`target(R)`) according to the following axiom:

```
target(R) :- guess(R,V), not sampled(R), V ≤ 80.
```

i.e., whenever the rock `R` is valuable with probability *lower than 80%*. This leads the agent to pick possibly invaluable rocks, resulting in a negative reward (-10). On the contrary, the rule learned from 40% of the traces only requires the rock to be at null distance from the agent:

$$\texttt{target(R) :- dist(R,D), not sampled(R), D} \leq \texttt{0.} \tag{24}$$

We remark that, as the grid size increases, the number of online simulations for POMCP (2^{15}) may not be sufficiently high to preserve performance with respect to pure POMCP (Figures 4b, 4c), especially when the learned specifications have significantly low quality. However, the deterioration in performance with specifications learned from 20% and 60% of the training traces is still more gradual than *POMCP + ASP bad* in Figures 8c-8d, where the return is very low even for small grid sizes.

In Figure 11 we show the results achieved by different policy heuristics in DESPOT (C++), with different number of rocks and on larger grid sizes. When *HIND* upper bound is used (Figures 11a, 11b), the discounted returns are comparable, confirming the important role of a good-quality upper bound for DESPOT performance. When the *TRIVIAL* upper bound is used (Figures 11c-11d), instead, the difference in policy heuristics is crucial to determine final performance of the algorithm, and matches the trend observed in Figure 10. In particular, results evidence that specifications learned from 60% of the original traces perform even worse than the ones learned from 20% of traces (this is also partly evident from Figure 10d, as the grid size increases). A possible reason lies in the higher number of `check(R)` actions executed with *ASP (60%)* lower bound. Exploiting the interpretability of logical specifications, from Appendix A we notice that, in this configuration, the following axiom holds:

```
check(R) :- target(R), guess(R,V), V ≤ 50.
```

This specification, combined with Equation (24), leads to more frequent grounding of `check(R)`. On the other hand, with *ASP (20%)* lower bound, the following axiom is learned:

```
check(R) :- target(R), guess(R,V), V ≤ 30.
```

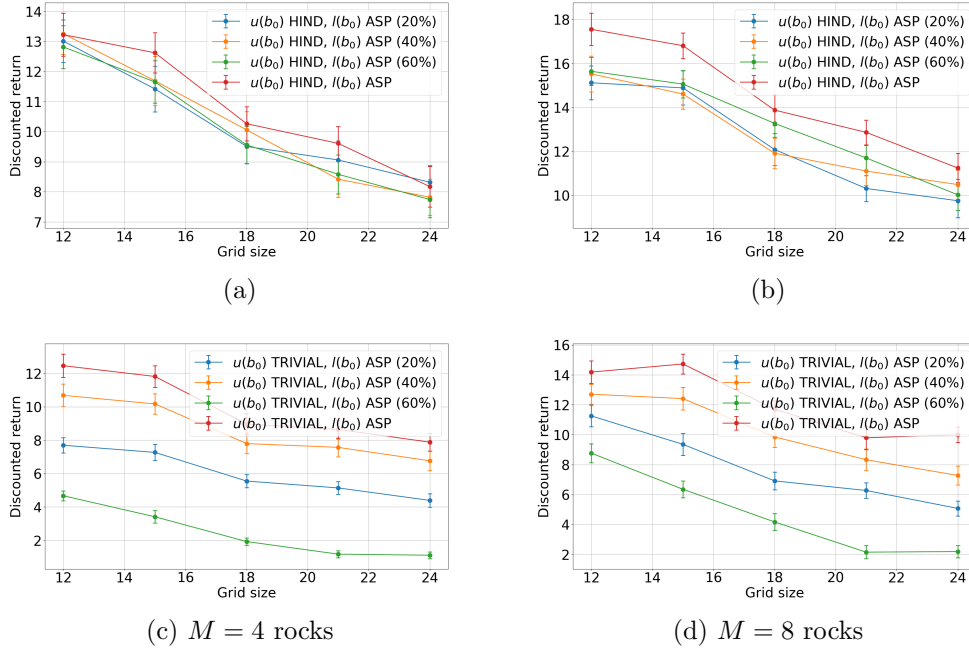which is more strict.

(a)

(b)

(c) $M = 4$ rocks

(d) $M = 8$ rocks

Figure 11: Results (mean ± standard deviation of the discounted return) for rocksample in DESPOT (C++) with policy heuristics learned from few examples (**EXP-4**). **Top**: Performance with hindsight (optimal) upper bound. **Bottom**: Performance with trivial upper bound.

We can conclude that, though the axioms converge to the best ones as the number of examples increases, the quality of learned policy heuristics for online POMDP planning may not follow the same trend. Hence, an interpretable formalism to understand the meaning of policy heuristics is desired, to identify potential fallacies and prevent unwanted behavior of the agent.

5.6 Comparison with Deep RL (EXP-5)

As mentioned in Section 2 and highlighted in Section 5.1, inductive logic programming for learning policy specifications increases interpretability and requires relatively few examples (1000 executions, though even 800 are enough for rocksample, from Table 4). In this section, we perform further quantitative experiments to highlight the benefits of our method with respect to deep RL approaches. We consider the work and code by Subramanian et al. (2022), where a deep neural architecture is adopted to efficiently train a Reinforcement Learning (RL) agent to solve the rocksample problem. Specifically, the authors outperform a state-of-the-art methodology based on proximal policy optimization and recurrent units, exploiting the approximate informativeness of the partially observable state (i.e., the so-called *Approximate Information State, AIS*[12]).

---

12. We use the maximum mean discrepancy to evaluate the informativeness of partially observable states, since it achieves the best results in rocksample in the original paper by Subramanian et al. (2022).
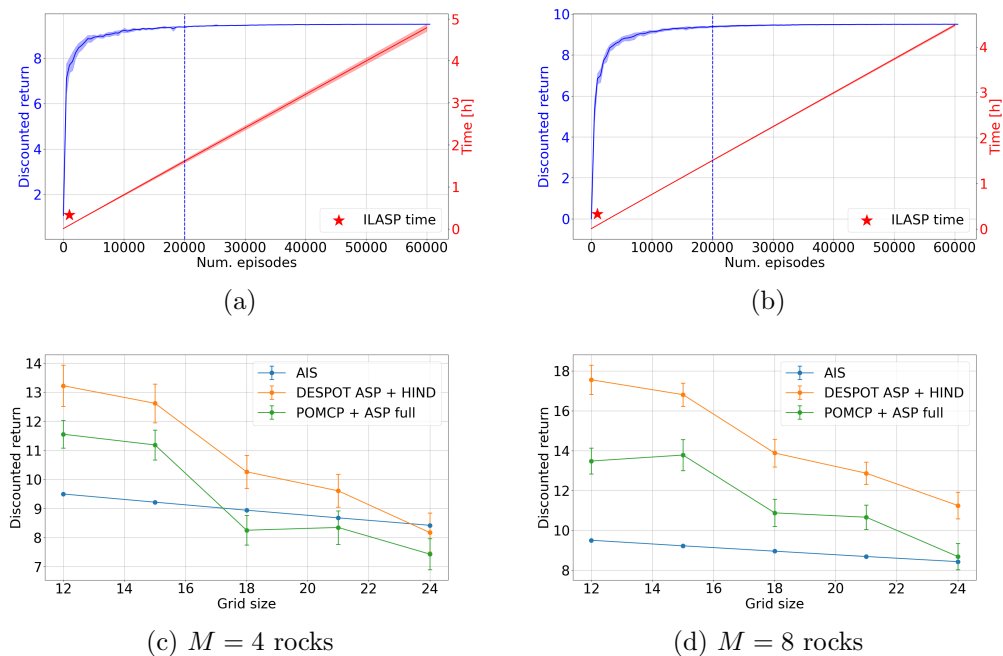
(a)

(b)

(c) $M = 4$ rocks

(d) $M = 8$ rocks

Figure 12: **EXP-5**: Comparison with black-box RL for rocksample. **Top**: Training return and time (mean ± standard deviation) on a 12×12 grid over 5 random seeds. **Bottom**: Inference return (mean ± standard deviation), training AIS for 20000 episodes.

Figures 12a-12b show the training performance of RL (*AIS*), in terms of training time and discounted return. Plots are obtained randomizing over 5 seeds. Since the RL method can generalize a learned policy to different grid sizes, but not to different number of rocks, we train it both with 4 (Figure 12a) and 8 rocks (Figure 12b), on a 12×12 grid. Approximately 20000 episodes of rocksample are required to achieve a stable discounted return, corresponding to ≈1.5 h training time (blue dashed lines in Figures 12a-12b). On the contrary, our learning methodology (*ILASP time*) requires only ≈0.3 h, including the time to generate 1000 traces with 4 rocks and the time to learn policy specifications from ILASP (see Section 5.1). Moreover, we can learn rules from a scenario with 4 rocks and generalize them to a scenario with more rocks. Our methodology is also clearly superior in terms of discounted return and generalization. In fact, from Figures 12c-12d, DESPOT (C++) and POMCP solvers equipped with learned logical specifications significantly outperform the performance of *AIS* on larger grid sizes (*POMCP + ASP full* achieves lower return only with 4 rocks with grid size $N \geq 18$, from Figure 12c).

6. Discussion

In this section we summarize the results of our experiments (with settings depicted in Table 1), highlighting the advantages of our pipeline for inductive learning of POMDP policy specifications and evidencing current limitations and potential directions to overcome them.

With reference to the objectives **EXP-1**-**EXP-5** stated in Section 5, we draw the following conclusions:

- *quality of learned policy heuristics* (**EXP-1**): we learned logical policy specifications for rocksample and pocman, starting from 1000 traces of execution (belief-action pairs) in small settings, namely, a 12×12 grid with 4 rocks in rocksample and a 10×10 map with 50% food probability and 2 mildly aggressive ghosts (75% chasing probability) in pocman. The traces correspond to POMCP executions achieving high discounted return, generated with 2^{15} online simulations and particles for belief approximation. The learned policy specifications were integrated in the UCT and rollout phases of POMCP, and used to compute the default policy and corresponding lower bound for DESPOT tree exploration. In POMCP, in the rocksample domain with the largest action space ($6 + M$ actions, being M the number of rocks), **policy heuristics significantly improve the performance of standard POMCP when the number of online simulations and particles is reduced** (up to 2^{10}). Our ablation study highlights that *performance increase when policy specifications are implemented in the rollout phase.* This introduces a significant computational cost; however, since a lower number of online simulations can be performed, **the overall cost of computing the best action at each time step is lower than POMCP without heuristics**. In pocman domain, POMCP performance do not significantly vary when heuristics are used. However, the lower bound based on logical specifications leads to performance (computational time per step and discounted return) comparable to optimal handcrafted heuristics by Silver and Veness (2010) in DESPOT.

- *Generality and scalability of learned heuristics* (**EXP-2**): we tested learned policy specifications in more challenging conditions for the two tasks, which were not present in the original 1000 traces. In particular, we increased the number of rocks (M) and the grid size (N) in rocksample, and modified the map size and ghost and food parameters in pocman. This increases the size of the action space and the planning horizon, i.e., the number of actions required to accomplish the task. In general, *learned heuristics achieve good performance also in these settings, proving their quality and generality even with very large actions space and long planning horizon* (see Table 1). Specifically, very good performance is achieved in rocksample with 20×20 grid and 20 rocks (corresponding to 26 available actions), where learned heuristics are computationally more efficient than handcrafted ones both in POMCP and DESPOT (requiring $\approx 40\% - 50\%$ time less to compute the best action at each step), while yielding an improvement with respect to pure POMCP and *TRIVIAL* lower bound. Finally, such results are confirmed when the heuristics are applied to state-of-the-art extensions of POMDP solvers, e.g., AdaOPS by Wu et al. (2021) which extends DESPOT, proving the cross-solver generality of learned policy specifications. **In pocman, learned heuristics perform much better than the *TRIVIAL* strategy and similarly to handcrafted, even when the task is significantly altered with respect to the training setting, requiring a different rationale to solve the task efficiently**. In fact, the large 17×19 map has 4 more aggressive ghosts (chasing probability $\rho_g = 100\%$, while in the training traces in Section 5.1 $\rho_g = 75\%$) and very little food (food probability $\rho_f = 20\%$, while in the training traces in Section

5.1 $\rho_g = 50\%$). This scenario is significantly different from the one used to generate the 1000 training traces, since the policy of the agent must account for the scarcity of food pellets (giving positive reward) and a more strict constraint to avoid ghosts. Moreover, the planning horizon is very long (on average 85 actions are executed by the agent). Unfortunately, it was not possible to perform pocman experiments in AdaOPS, since this domain is not implemented in the publicly available Julia package. Nevertheless, we believe that rocksample results are sufficiently convincing, since it is the task with the largest action space (spanning from 10 actions with 4 rocks to 26 actions with 20 rocks) and the planning horizon reaches 67 actions when $N = M = 20$, and 47 actions when $N = 24, M = 8$.

- *Effect of low quality policy specifications* (**EXP-3**)[13]: in the rocksample domain, we generated traces for learning logical policy specifications, using POMCP with 2^{10} online simulations and particles. This corresponds to the value where the discounted return achieved by *POMCP + ASP full* significantly deteriorates (see Figure 4). In these conditions, new policy specifications in Equation (23) are learned, which contain low quality information about the task. Specifically, the criterion for selecting target rocks (`target(R)`) and for exiting the grid (`exit`) is different. In DESPOT, this deteriorates the performance of the online solver. However, **in POMCP, thanks to the *soft policy guidance* approach adopted in the rollout phase instead of action pruning (see Section 4.4.1), the achieved discounted return is not severely affected, because the optimality of the algorithm with many particles and online simulations is preserved**. This experiment also evidences the effect of considering a non exhaustive set of environmental features to learn logical specifications. In fact, learned specifications exclude some environmental atoms, with respect to the best axioms learned with 2^{15} simulations. For instance, the specification for `target(R)` includes only information about the distance between rocks and the agent, ignoring the probability of rock value. Conversely, the specification for `exit` does not depend on the relative distance to unsampled rocks.

- *Effect of the number of training traces* (**EXP-4**): in the rocksample domain, we varied the number of traces (hence, generated examples for ILASP), considering different percentages of them ranging from 20% to 80 %. This experiment evidences that **ILASP is able to converge to the best rules (from 1000 traces) as the number of examples increases**, progressively reducing the average distance from them (see Table 4). Furthermore, this experiment strenghtens the results achieved with **EXP-3**, since learned rules often have very low quality (e.g., `target(R)` selects rocks with low value probability, see Appendix A) and depend only on a subset of the environmental features. Nevertheless, the performance of *POMCP + ASP full* is still acceptable, at least for POMCP when the number of online simulations and particles is sufficiently high and the grid size does not increase much ($N \leq 18$). With fewer online simulations or in larger grids asymptotic optimality of POMCP cannot be guaranteed anymore, since bad policy heuristics may require significantly many online simulations. Hence, the performance significantly deteriorates when policy specifications are not good, as

---

13. **EXP-3**-**EXP-4** are performed only for rocksample, since the learned specifications for pocman are not affected when either the quality or the number of traces change.

for DESPOT. However, **the interpretability of logical policy specifications allows to identify potential fallacies in the learned axioms, with the potential to prevent unwanted behavior of the agent**.

- *Comparison with black-box architectures (deep RL)* (**EXP-5**): we compare our methodology with the work by Subramanian et al. (2022), a state-of-the-art methodology based on approximate information state exploration and the use of autoencoders and recurrent neural units to learn the best strategy to solve the rocksample domain. Our methodology has three fundamental advantages, compared to training an agent with deep RL. First, it has **higher data and computational efficiency**, requiring only 1000 example executions (for the rocksample task) vs. 20000 executions, and ≈ 0.3 h vs. ≈ 1.5 h. Second, **the discounted return is much higher with learned heuristics**, since POMCP and DESPOT equipped with logical specifications are mostly superior to the neural method. We believe that one reason for this lies in the use of the POMDP model (e.g., the transition model) in online solving with POMCP, DESPOT and AdaOPS, which is not available to a neural network. Third, **logical specifications are more interpretable**. This allows to identify potential fallacies in learned specifications, e.g., it is useful in **EXP-4** to explain the lower performance achieved by specifications learned from 20% and 60% of the training traces. Hence, logical policy heuristics increase the overall transparency of the decision making process.

Our empirical analysis also evidenced potential directions of improvement of our methodology. Specifically, the following open problems remain to be addressed:

- *Definition of complete environmental features*: in our experiments, we assume that environmental features representing the POMDP belief are available from the transition and reward maps. This holds for many POMDP problems, thus, in general, the problem of defining these features is easier than manually defining policy specifications, since the policy is clearly unknown in POMDPs. Moreover, **EXP-3**-**EXP-4** show that our approach to exploit logical specifications in POMCP is robust to the unavailability of some features, yielding a moderate degradation of performance. However, in very complex real-world domains, e.g., robotic systems (Zuccotto et al., 2022b; Castellini et al., 2021; Pajarinen et al., 2022) and environmental exploration (Thompson & Guihen, 2019; Steccanella et al., 2020; Bravo et al., 2019), it may not be easy to extract all features from the definition of the POMDP problem, which may be partly incomplete. Moreover, in the rocksample domain we introduced the `target(R)` atom, which is a useful high-level concept for the task, but cannot be directly retrieved from the transition and reward maps. A possible solution to this problem is to exploit other algorithms for ILP, e.g., bottom-up meta-interpretative learning by Hocquette and Muggleton (2021) and Popper by Cropper and Morel (2021), which support predicate invention. This helps discover relevant concepts about the planning domain directly from examples. However, usually these algorithms do not support the full expressiveness of an action language for planning, e.g., weak constraints for optimal decision making in ASP. Hence, a tradeoff must be reached.

- *Expressiveness of logical specifications*: we showed how to learn logical specifications for actions from examples of execution. However, our rules only consider the current situation of the task, e.g., the current observable state and belief, and provide suggestions to the POMDP solver only about the next action to be executed. In complex tasks involving, e.g., long planning horizons or non-trivial interaction with the environment, it may be useful to learn *temporal policy heuristics*. However, the problem of learning temporal logic axioms is still an open challenge. In fact, typically only specific fragments of temporal logics are learned, e.g., in the event calculus paradigm (Meli et al., 2020, 2021a), or propositional temporal rules can be extracted from examples exploiting graph structure learning (De Giacomo et al., 2020). Hence, extension to temporal first order logic and the full ASP expressiveness still have to be investigated from a theoretical perspective. We believe that combining temporal expressiveness with predicate invention could improve the quality of learned policy heuristics and their impact on online POMDP planning, allowing to emulate the performance of handcrafted heuristics by Silver and Veness (2010), e.g., considering the number of times a rock has been checked in rocksample (see Section 5.1).

- *Online learning of policy specifications*: our methodology allows to learn policy specifications from traces of execution offline. In the future, it will be interesting to investigate *online learning* of such specifications, as the agent acquires experience from its interaction with the environment. This has the potential to refine and improve agent's performance in more challenging task scenarios. It could be achieved by, e.g., extending state-of-the-art approaches for online learning of ASP rewards (Furelos-Blanco et al., 2021, currently focusing on fully observable scenarios, i.e., MDPs) with fast online ILP techniques, such as incremental ILASP from streams of data (Law et al., 2022).

7. Conclusion

State-of-the-art online POMDP planners require good task-specific policy heuristics to achieve good performance in complex environments involving many actions or long planning horizons. In this paper, we proposed a methodology based on ILP to learn ASP policy specifications from POMDP traces of execution generated by *any solver without heuristics*. We integrate learned heuristics in different state-of-the-art online POMDP planners, specifically for biasing rollout and action values in POMCP, and to define the default policy and lower bound for state value in DESPOT and AdaOPS. In this way, we achieve similar performance as optimal handcrafted heuristics in two challenging benchmark domains: rocksample, involving many actions and potentially long planning horizon; and pocman, with very long planning horizon. Moreover, learned heuristics help reduce the time for optimal action computation when the action space is large, and can be learned from small domains and efficiently generalize to larger more challenging problem instances (e.g., more rocks and larger grid sizes in rocksample, larger maps and more numerous and aggressive ghosts in pocman with fewer food pellets). Also, the use of logic programming allows to learn *interpretable heuristics* which provide some explanation of the decision-making process of the POMDP agent, significantly outperforming a state-of-the-art methodology based on autoencoders and recurrent neural units in terms of *training data and computational efficiency*

and *achieved discounted return.* We achieve this by integrating commonsense environmental semantic features in our pipeline, which are related to domain knowledge concerning the basic POMDP problem definition (the transition map), hence can be defined by a user with basic domain knowledge. Crucially, adopting a *soft policy guidance* approach in POMCP, the planner is robust to bad heuristics learned, e.g., from low-quality and few training traces or when some semantic features are missing.

This paper is a first attempt to learn heuristics for POMDP solvers from execution traces. As such, it paves the way towards several interesting research directions. In the future, we plan to investigate extensions to temporal logic and predicate invention, to increase the quality of policy specifications and online planning performance. Moreover, we will investigate more challenging real-world domains, e.g., robotic domains, and implement an ILP methodology for online learning and refinement of policy heuristics from experience.

Acknowledgments

Appendix A. Learned Policy Specifications for Rocksample in EXP-4

Here, we report learned policy specifications for rocksample, starting from different subsets of the original 1000 training traces used in Section 5.1. Specifically, we consider subsets containing $\{20, 40, 60, 80\}\%$ of the original traces, hence corresponding CDPIs. We do not report learned specifications from 80% of the total examples, since they correspond to the best policy heuristics in Equation (21).

A.1 Policy Heuristics from 20% of the Original Traces

The following specifications are learned from this subset of traces / examples.

```
east :- target(R), delta_x(R,D), D ≥ 1, D ≤ 2.
west :- target(R), delta_x(R,D), D ≤ -1.
north :- target(R), delta_y(R,D), D ≥ 1.
south :- target(R), delta_y(R,D), D == -2.
target(R) :- dist(R,D), not sampled(R), D ≤ 0.
target(R) :- guess(R,V), not sampled(R), V ≤ 80.
check(R) :- guess(R,V), target(R), V ≤ 30.
sample(R) :- target(R), dist(R,D), D ≤ 0, not sampled(R), guess(R,V), V ≥ 90.
exit :- num_sampled(N), N ≥ 25.
```

A.2 Policy Heuristics from 40% of the Original Traces

The following specifications are learned from this subset of traces / examples.

```
east :- target(R), delta_x(R,D), D ≥ 1.
west :- target(R), delta_x(R,D), D ≤ -1.
north :- target(R), delta_y(R,D), D ≥ 2.
south :- target(R), delta_x(R,D), D ≤ 0.
target(R) :- dist(R,D), not sampled(R), D ≤ 0.
check(R) :- guess(R,V), target(R), V ≤ 50.
check(R) :- guess(R,V), dist(R,D), not sampled(R), V ≤ 80, D ≤ 0.
sample(R) :- target(R), dist(R,D), D ≤ 0, not sampled(R), guess(R,V), V ≥ 90.
exit :- num_sampled(N), N ≥ 25, guess(R,V), V ≥ 80, not sampled(R).
```

A.3 Policy Heuristics from 60% of the Original Traces

The following specifications are learned from this subset of traces / examples.

```
east :- target(R), delta_x(R,D), D ≥ 1.
west :- target(R), delta_x(R,D), D ≤ -1.
north :- target(R), delta_y(R,D), D ≥ 1.
south :- target(R), delta_y(R,D), D == -2.
target(R) :- guess(R,V), not sampled(R), V ≤ 80.
check(R) :- guess(R,V), target(R), V ≤ 50.
check(R) :- guess(R,V), dist(R,D), not sampled(R), V ≤ 80, D ≤ 0.
sample(R) :- target(R), dist(R,D), D ≤ 0, not sampled(R), guess(R,V), V ≥ 90.
exit :- num_sampled(N), N ≥ 25.
```

References

Amini, S., Palhang, M., & Mozayani, N. (2023). POMCP-based decentralized spatial task allocation algorithms for partially observable environments. *Applied Intelligence*, *53*(10), 12613–12631.

Andriushchenko, R., Češka, M., Junges, S., & Katoen, J.-P. (2022). Inductive synthesis of finite-state controllers for POMDPs. In *Uncertainty in Artificial Intelligence*, pp. 85–95. PMLR.

Balduccini, M. (2007). Learning Action Descriptions with A-Prolog: Action Language C. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*.

Basu, S., Rajesh, S., Zheng, K., Tellex, S., & Bahar, R. I. (2021). Parallelizing POMCP to solve complex POMDPs. In *Rss workshop on software tools for real-time optimal control*.

Bhattacharya, S., Kailas, S., Badyal, S., Gil, S., & Bertsekas, D. (2021). Multiagent rollout and policy iteration for POMDP with application to multi-robot repair problems. In *Conference on Robot Learning*, pp. 1814–1828. PMLR.

Bonet, B., & Geffner, H. (2020). Learning First-Order Symbolic Representations for Planning from the Structure of the State Space. In *ECAI 2020*, pp. 2322–2329. IOS Press.

Bravo, R. Z. B., Leiras, A., & Cyrino Oliveira, F. L. (2019). The use of UAV s in humanitarian relief: An application of POMDP-based methodology for finding victims. *Production and Operations Management*, *28*(2), 421–440.

Cai, P., & Hsu, D. (2022). Closing the Planning–Learning Loop With Application to Autonomous Driving. *IEEE Transactions on Robotics*, *39*(2), 998–1011.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., & Schaub, T. (2020). ASP-Core-2 input language format. *Theory and Practice of Logic Programming*, *20*(2), 294–309.

Cassandra, A., Littman, M. L., & Zhang, N. L. (1997). Incremental pruning: a simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pp. 54–61.

Cassandra, A. R. (1998). A survey of POMDP applications. In *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, Vol. 1724.

Castellini, A., Chalkiadakis, G., & Farinelli, A. (2019). Influence of State-Variable Constraints on Partially Observable Monte Carlo Planning. In *IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 5540–5546. ijcai.org.

Castellini, A., Marchesini, E., & Farinelli, A. (2021). Partially Observable Monte Carlo Planning with state variable constraints for mobile robot navigation. *Engineering Applications of Artificial Intelligence*, *104*, 104382.

Cheng, C.-A., Kolobov, A., & Swaminathan, A. (2021). Heuristic-guided reinforcement learning. *Advances in Neural Information Processing Systems*, *34*, 13550–13563.

Cropper, A., & Morel, R. (2021). Learning programs by learning from failures. *Machine Learning*, *110*, 801–856.

Danesh, M. H., Cai, P., & Hsu, D. (2023). LEADER: Learning Attention over Driving Behaviors for Planning under Uncertainty. In *Conference on Robot Learning*, pp. 199–211. PMLR.

D'Asaro, F. A., Spezialetti, M., Raggioli, L., & Rossi, S. (2020). 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*, Vol. 17, pp. 855–859.

De Giacomo, G., Favorito, M., Iocchi, L., & Patrizi, F. (2020). Imitation learning over heterogeneous agents with restraining bolts. In *Proceedings of the international conference on automated planning and scheduling*, Vol. 30, pp. 517–521.

De Giacomo, G., Iocchi, L., Favorito, M., & Patrizi, F. (2019). Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In *Proceedings of the international conference on automated planning and scheduling*, Vol. 29, pp. 128–136.

De la Rosa, T., Jiménez, S., Fuentetaja, R., & Borrajo, D. (2011). Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence Research*, *40*, 767–813.

Erdem, E., & Patoglu, V. (2018). Applications of ASP in robotics. *KI-Künstliche Intelligenz*, *32*(2-3), 143–149.

Furelos-Blanco, D., Law, M., Jonsson, A., Broda, K., & Russo, A. (2021). Induction and exploitation of subgoal automata for reinforcement learning. *Journal of Artificial Intelligence Research*, *70*, 1031–1116.

Gil, Y. (1994). Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. In *International Conference on Machine Learning*, pp. 87–95 New Brunswick, USA.

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

Giuliari, F., Castellini, A., Berra, R., Del Bue, A., Farinelli, A., Cristani, M., Setti, F., & Wang, Y. (2021). POMP++: Pomcp-based Active Visual Search in unknown indoor environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1523–1530. IEEE.

Gupta, H., Hayes, B., & Sunberg, Z. (2022). Intention-Aware Navigation in Crowds with Extended-Space POMDP Planning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pp. 562–570.

Hocquette, C., & Muggleton, S. H. (2021). Complete bottom-up predicate invention in meta-interpretive learning. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 2312–2318.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and Acting in Partially Observable Stochastic Domains. *Artif. Intell.*, *101*(1–2), 99–134.

Kambhampati, S., Sreedharan, S., Verma, M., Zha, Y., & Guan, L. (2022). Symbols as a lingua franca for bridging human-ai chasm for explainable and advisable ai systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36, pp. 12262–12267.

Katt, S., Oliehoek, F. A., & Amato, C. (2017). Learning in POMDPs with Monte Carlo tree search. In *International Conference on Machine Learning*, pp. 1819–1827. PMLR.

Kocsis, L., & Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. In *Proc. ECML'06*, pp. 282–293 Berlin, Heidelberg. Springer-Verlag.

Law, M. (2023). Conflict-driven inductive logic programming. *Theory and Practice of Logic Programming*, *23*(2), 387–414.

Law, M., Broda, K., & Russo, A. (2022). Search Space Expansion for Efficient Incremental Inductive Logic Programming from Streamed Data. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 2697–2704. International Joint Conferences on Artificial Intelligence Organization.

Law, M., Russo, A., & Broda, K. (2016). Iterative learning of answer set programs from context dependent examples. *Theory and Practice of Logic Programming*, *16*(5-6), 834–848.

Law, M., Russo, A., & Broda, K. (2018). The complexity and generality of learning answer set programs. *Artificial Intelligence*, *259*, 110–146.

Lee, Y., Cai, P., & Hsu, D. (2021). MAGIC: Learning Macro-Actions for Online POMDP Planning . In *Proceedings of Robotics: Science and Systems* Virtual.

Leonetti, M., Iocchi, L., & Patrizi, F. (2012). Automatic generation and learning of finite-state controllers. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pp. 135–144. Springer.

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

Lifschitz, V. (1999). Answer set planning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 373–374. Springer.

Maliah, S., Komarnitski, R., & Shani, G. (2022). Computing contingent plan graphs using online planning. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, *16*(1), 1–30.

Marques-Silva, J., Lynce, I., & Malik, S. (2021). Conflict-driven clause learning SAT solvers. In *Handbook of satisfiability*, pp. 133–182. IOS press.

Mazzi, G., Castellini, A., & Farinelli, A. (2021a). 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)*, p. 889–897. International Foundation for Autonomous Agents and Multiagent Systems.

Mazzi, G., Castellini, A., & Farinelli, A. (2021b). Rule-based Shielding for Partially Observable Monte-Carlo Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, Vol. 31, pp. 243–251.

Mazzi, G., Castellini, A., & Farinelli, A. (2023a). Risk-aware shielding of Partially Observable Monte Carlo Planning policies. *Artificial Intelligence*, *324*, 103987.

Mazzi, G., Meli, D., Castellini, A., & Farinelli, A. (2023b). Learning Logic Specifications for Soft Policy Guidance in POMCP. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pp. 373–381.

Meli, D., Fiorini, P., & Sridharan, M. (2020). Towards inductive learning of surgical task knowledge: A preliminary case study of the peg transfer task. *Procedia Computer Science*, *176*, 440–449.

Meli, D., Nakawala, H., & Fiorini, P. (2023). Logic programming for deliberative robotic task planning. *Artificial Intelligence Review*, *56*(6), 9011–9049.

Meli, D., Sridharan, M., & Fiorini, P. (2021a). Inductive learning of answer set programs for autonomous surgical task planning. *Machine Learning*, *110*(7), 1739–1763.

Meli, D., Tagliabue, E., Dall'Alba, D., & Fiorini, P. (2021b). Autonomous tissue retraction with a biomechanically informed logic based framework. In *2021 International Symposium on Medical Robotics (ISMR)*, pp. 1–7. IEEE.

Mota, T., & Sridharan, M. (2020). Axiom Learning and Belief Tracing for Transparent Decision Making in Robotics. In *AAAI Fall Symposium on Artificial Intelligence for Human-Robot Interaction: Trust and Explainability in Artificial Intelligence for Human-Robot Interaction*.

Muggleton, S. (1991). Inductive logic programming. *New generation computing*, *8*(4), 295–318.

Pajarinen, J., Lundell, J., & Kyrki, V. (2022). POMDP planning under object composition uncertainty: Application to robotic manipulation. *IEEE Transactions on Robotics*, *39*(1), 41–56.

Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of operations research*, *12*(3), 441–450.

Paquet, S., Tobin, L., & Chaib-Draa, B. (2005). An online POMDP algorithm for complex multiagent environments. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 970–977.

Rabold, J., Siebers, M., & Schmid, U. (2018). 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*, pp. 105–117. Springer.

Rodriguez, I. D., Bonet, B., Romero, J., & Geffner, H. (2021). Learning First-Order Representations for Planning from Black Box States: New Results. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, Vol. 18, pp. 539–548.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, *0*, 0.

Shani, G. (2013). Task-based decomposition of factored POMDPs. *IEEE transactions on cybernetics*, *44*(2), 208–216.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, *362*(6419), 1140–1144.

Silver, D., & Veness, J. (2010). Monte-Carlo planning in large POMDPs. *Advances in neural information processing systems*, *23*.

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

Sridharan, M., & Meadows, B. (2018). Knowledge Representation and Interactive Learning of Domain Knowledge for Human-Robot Collaboration. *Advances in Cognitive Systems*, *7*, 77–96.

Steccanella, L., Bloisi, D., Castellini, A., & Farinelli, A. (2020). Waterline and Obstacle Detection in Images from Low-Cost Autonomous Boats for Environmental Monitoring. *Robot. Auton. Syst.*, *124*(C).

Subramanian, J., Sinha, A., Seraj, R., & Mahajan, A. (2022). Approximate information state for approximate planning and reinforcement learning in partially observed systems. *The Journal of Machine Learning Research*, *23*(1), 483–565.

Sunberg, Z., & Kochenderfer, M. (2018). Online algorithms for POMDPs with continuous state, action, and observation spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 28, pp. 259–263.

Tagliabue, E., Meli, D., Dall'Alba, D., & Fiorini, P. (2022). Deliberation in autonomous robotic surgery: a framework for handling anatomical uncertainty. In *2022 IEEE International Conference on Robotics and Automation (ICRA 2022)*.

Thompson, F., & Guihen, D. (2019). Review of mission planning for autonomous marine vehicle fleets. *Journal of Field Robotics*, *36*(2), 333–354.

Toyer, S., Thiébaux, S., Trevizan, F., & Xie, L. (2020). Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, *68*, 1–68.

Tran, S. C., Pontelli, E., Balduccini, M., & Schaub, T. (2023). Answer Set Planning: A Survey. *Theory and Practice of Logic Programming*, *23*(1), 226–298.

Veronese, C., Meli, D., Bistaffa, F., Rodríguez Soto, M., Farinelli, A., & Rodríguez-Aguilar, J. A. (2023). Inductive Logic Programming For Transparent Alignment With Multiple Moral Values.. 2nd International Workshop on Emerging Ethical Aspects of AI (BEWARE-23).

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*(7782), 350–354.

Wang, Y., Giuliari, F., Berra, R., Castellini, A., Del Bue, A., Farinelli, A., Cristani, M., & Setti, F. (2021a). POMP: Pomcp-based Online Motion Planning for active visual search in indoor environments. In *Proc. of British Machine Vision Conference (BMVC)*. BMVA Press.

Wang, Y., Newaz, A. A. R., Hernández, J. D., Chaudhuri, S., & Kavraki, L. E. (2021b). Online partial conditional plan synthesis for POMDPs with safe-reachability objectives: Methods and experiments. *IEEE Transactions on Automation Science and Engineering*, *18*(3), 932–945.

Wu, C., Yang, G., Zhang, Z., Yu, Y., Li, D., Liu, W., & Hao, J. (2021). Adaptive online packing-guided search for POMDPs. *Advances in Neural Information Processing Systems*, *34*, 28419–28430.

Ye, N., Somani, A., Hsu, D., & Lee, W. S. (2017). DESPOT: Online POMDP planning with regularization. *Journal of Artificial Intelligence Research, 58*, 231–266.

Yoon, S., Fern, A., Givan, R., & Kambhampati, S. (2008). Probabilistic planning via determinization in hindsight. In *Proceedings of the 23rd national conference on Artificial intelligence-Volume 2*, pp. 1010–1016.

Zuccotto, M., Castellini, A., & Farinelli, A. (2022a). Learning State-Variable Relationships for Improving POMCP Performance. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, SAC*, pp. 739–747. Association for Computing Machinery.

Zuccotto, M., Piccinelli, M., Castellini, A., Marchesini, E., & Farinelli, A. (2022b). Learning State-Variable Relationships in POMCP: A Framework for Mobile Robots. *Frontiers in Robotics and AI, 9*.