Online Inductive Learning from Answer Sets for Efficient Reinforcement Learning Exploration

Celeste Veronese(✉), Daniele Meli, and Alessandro Farinelli

Department of Computer Science, University of Verona, Verona, Italy
{celeste.veronese,daniele.meli,alessandro.farinelli}@univr.it

Abstract. This paper presents a novel approach combining inductive logic programming with reinforcement learning to improve training performance and explainability. We exploit inductive learning of answer set programs from noisy examples to learn a set of logical rules representing an explainable approximation of the agent's policy at each batch of experience. We then perform answer set reasoning on the learned rules to guide the exploration of the learning agent at the next batch, without requiring inefficient reward shaping and preserving optimality with soft bias. The entire procedure is conducted during the online execution of the reinforcement learning algorithm. We preliminarily validate the efficacy of our approach by integrating it into the Q-learning algorithm for the Pac-Man scenario in two maps of increasing complexity. Our methodology produces a significant boost in the discounted return achieved by the agent, even in the first batches of training. Moreover, inductive learning does not compromise the computational time required by Q-learning and learned rules quickly converge to an explanation of the agent's policy.

Keywords: Inductive Logic Programming · Neurosymbolic AI · Reinforcement Learning · Explainable AI · Answer Set Programming

1 Introduction

The intersection of symbolic, logic-based reasoning and Reinforcement Learning (RL) represents a trending topic in Artificial Intelligence (AI) research, aiming to harness the strengths of both domains and address their individual limitations [19]. Current state-of-the-art approaches in model-free RL, in fact, predominantly rely on extensive data or pre-defined environmental models, facing significant challenges in terms of efficiency and scalability. Furthermore, the decision process underlying policy generation is not transparent to other agents and humans, hindering certifiability and trustworthiness. On the other hand, symbolic AI works well in the small data regime but does not perform well on non-symbolic data and is not noise tolerant [1]. Incorporating symbolic and logical formalisms into RL systems, as highlighted by [20], can significantly boost their

P. Bruno et al. (Eds.): HYDRA 2024, CCIS 2492, pp. 93–106, 2025.
https://doi.org/10.1007/978-3-031-89366-7_7

interpretability, thereby fostering wider acceptance and diffusion while also playing a crucial role in improving the policy and the training phase. However, symbolic learning and reasoning may significantly increase the computational burden of RL, thus limiting the effective real-time integration of neurosymbolic approaches [8] or the scalability to non-trivial tasks, e.g., where multiple predicates and scopes for logical variables shall be defined [7].

Inspired by results obtained by [4,13] in model-based RL, we present a novel approach that exploits Inductive Logic Programming (ILP, [14]) to iteratively learn human-interpretable policy heuristics *online*, during the training of RL agents, and directly use them to bias the agent in its exploration process. Specifically, for each batch of experiences (state-action sequences) collected by the RL agent during training, we rank them by the cumulative return and convert them to a logical formalism to build examples for ILP. This representation maps states and actions to basic human-level concepts about the task, thus enhancing the interpretability of the trained policy. ILP then learns a logical approximation of the policy, which is then used in the following batch to bias the agent's exploration to improve the performance by avoiding wrong or dangerous actions. We adopt the logical formalism of Answer Set Programming (ASP) [15] to represent logical specifications and perform online heuristic reasoning. ASP is chosen as it can be considered the state-of-the-art in planning domain representation for autonomous agents [6]. In this way, we can rely on the latest developments in scalable and efficient ILP, even in the presence of many examples [16]. We preliminarily apply our methodology on the simple approximate Q-learning algorithm [5], to highlight better the variations introduced by our approach. However, any RL algorithm that includes a random exploration phase could be used (e.g. Double Deep Q-Network [3]). In summary, this paper presents the following contributions to the state-of-the-art:

- we introduce an innovative approach to online neurosymbolic learning and reasoning. Our methodology does not simply shape the reward of the agent according to the learned logical heuristics, but directly guides RL exploration, resulting in a tighter and more efficient neurosymbolic interconnection [19]. Moreover, we preserve RL optimality guarantees adopting a probabilistic soft bias approach;
- differently from [13], we perform the learning of informative logical heuristics *online* during RL training while maintaining the expressiveness of ASP. To this aim, we exploit fast scalable ILP as in [16];
- we empirically evaluate the effectiveness of our methodology in the benchmark Pac-Man scenario on maps of increasing complexity with contrastive agents. We obtain a significant boost in the discounted return achieved by the agent. Furthermore, while the inductive learning component minimally affects computational time, it facilitates a swift convergence of learned rules, leading to a coherent and comprehensive explanation of the underlying black-box policy.

2 Related Work

Recent works demonstrate that incorporating existing knowledge into RL and Markov Decision Processes (MDPs) can greatly enhance the development of effective policies [19]. The integration of such knowledge through logical formalisms has significantly improved policy computation. For instance, the REBA framework by [21] employs ASP to define spatial relations in a domestic setting, guiding a robotic agent to select specific rooms for inspection while addressing more straightforward MDP tasks locally. Similarly, DARLING [27] uses ASP to restrict MDP exploration in simulated grid environments and real-world robotics applications. Furthermore, [22,23] utilize linear temporal logic to direct exploration in MDPs. Logical constraints also play a crucial role in avoiding undesirable behaviours, particularly in safety-sensitive scenarios [24,25]. However, since logical heuristics usually express policy-related knowledge (unknown to the agent), they need to be defined by expert users. In the alternative, a huge amount of high-quality training examples are needed to learn them, such as in [26], in which temporal logic specifications are learnt as finite automata from good example traces and used to shape the reward signal. For this reason, attempts have been made to combine neural and symbolic learning. For instance, in [7] Neural Markov Logic Networks are used to learn relational representations from structured examples. However, the approach is computationally inefficient out of very simple domains, involving few variables and predicates. In [28] a deep relational reinforcement learning approach to learn generalizable policies is proposed. Nonetheless, the agent only adopts the logical policy at training convergence, thus not fully exploiting the synergy with neural methods, which are inherently more noise-robust and can help refine the agent's performance. Furthermore, even the solution presented in [28] suffers from poor scalability, requiring task-specific constraints on the available search space for logical policies.

In this paper, we combine deep RL with an ILP framework under the ASP semantics, which offers great expressiveness for structured task representation and reasoning in a fragment of first-order logic [6]. ILP under the ASP semantics has been proven successful in numerous scenarios, e.g., in enhancing the explainability of black-box models [29,30] and gaining task knowledge [9,31]. Among different available implementations [11,12], we adopt the popular FastLAS system [16], which can scale to very large search spaces. In this way, we can efficiently learn ASP policy approximations online while training the RL agent, solving the computational limitations posed by [7,28].

Our work is close to the one in [8], where ASP rules are learned to define reward machines in RL settings. However, our solution is more scalable thanks to the FastLAS approach. Furthermore, we do not use rules to shape the reward signal of the RL agent, which may be inefficient in effectively boosting the performance of RL training since still many interactions with the environment may be required to learn the value of action [19]. Instead, we take inspiration from [13] and perform ASP reasoning over the learned policy heuristics to bias the exploration of an RL agent with a higher probability towards heuristic-suggested actions.

3 Background

We now introduce some basic notions about the approximate Q-Learning algorithm, ASP and ILP, and describe the benchmark domain we used to test our methodology.

3.1 Approximate Q-Learning

Q-learning is a model-free reinforcement learning algorithm used to find an optimal action-selection policy [5]. The goal of the agent is to learn the optimal policy that maximizes the expected cumulative reward over time. The core concept in Q-learning is the action-value function, $Q(s, a)$, which estimates the expected future rewards for taking action a in state s. The Q-learning update rule is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$$

where s_t is the current state, a_t is the action taken in that state, r_{t+1} is the reward received after taking action a_t, and s_{t+1} is the next state. The parameter $\alpha \in [0, 1]$ is the learning rate, which controls how much new information overrides old information, and $\gamma \in [0, 1]$ is the discount factor, which determines the importance of future rewards. The algorithm converges to the optimal action-value function $Q^*(s, a)$ under certain conditions, such as a decaying learning rate and sufficient exploration of all state-action pairs. The optimal policy $\pi^*(s)$ is then derived by choosing the action that maximizes the Q-value in each state:

$$\pi^*(s) = \arg\max_a Q^*(s, a).$$

Approximate Q-learning addresses scalability issues inherent in traditional Q-learning, particularly in environments with large or continuous state spaces [5]. The key idea is to approximate the Q-function using a combination of features instead of the full state. That is, instead of recording everything in detail, we think about what is most important to know, and model that.

3.2 Answer Set Programming

Answer Set Programming (ASP) defines a domain as a set of logical statements (axioms) articulating the logical relationships between entities represented as predicates and variables (atoms) [18]. Axioms considered in this work are *normal rules* $\mathtt{h}$:- $\mathtt{b_1}, \ldots, \mathtt{b_n}$, which define the body of the rule (i.e. the logical conjunction of literals $\bigwedge_{\mathtt{i}=1}^{\mathtt{n}} \mathtt{b_i}$) as a precondition for the head $\mathtt{h}$. We say that a variable is grounded when assigned a particular value. Consequently, an atom is grounded when all its variables are grounded. Given an ASP program P, its Herbrand base $\mathcal{H}(P)$ defines the set of ground atoms which can be generated from it. From an ASP domain definition, an ASP solver computes the *answer sets*, i.e., the minimal set of literals that satisfies the given logic program according

to the stable model semantics. This work assumes that body atoms represent environmental features describing S in the MDP, while head atoms are actions. Answer sets will then contain feasible actions for the RL agent.

3.3 Inductive Logic Programming

A generic ILP task under a logical formalism F [14] is defined as a tuple $\mathcal{T} = \langle B, S_M, E \rangle$, consisting of background knowledge B expressed in a logic formalism F, a search space S_M containing the set of possible axioms to be learned (defined, e.g., via a mode declaration M [33]), and a set of examples E, all expressed in the syntax of F. The goal is to find a hypothesis (i.e. an axiom) $H \subseteq S_M$ covering E. Under the ASP semantics, we consider the generic case where examples are *weighted context dependent partial interpretations* (WCDPI's) [32]. A partial interpretation e^{pi} is a pair of sets of ground atoms $\langle e^{inc}, e^{exc} \rangle$. An interpretation (i.e., a set of ground atoms) I extends e iff $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$. A WCDPI is then a tuple $e = \langle e^{id}, e^{pen}, e^{pi}, e^{ctx} \rangle$, where e^{id} is an identifier for e, e^{pen} is either a positive integer or ∞, called a penalty, e^{pi} is a partial interpretation, and e^{ctx} is an ASP program called the context. A WCDPI e is accepted by a program P iff there is an answer set of $P \cup e^{ctx}$ that extends e^{pi}. Following the definition by [32], the goal of ILP is then to find a hypothesis $H \subseteq S_M$ with minimal length (i.e., number of atoms) and $\sum_e e^{pen}$, for all examples e not accepted by $H \cup B$. Since we are interested in discovering normal rules matching actions to environmental features, e^{inc}, e^{exc} represent executed and not executed actions, respectively, while e^{ctx} is the set of ground environmental features. We employ FastLAS learner by [16] for fast computation from acquired examples gathered from RL batches of experience.

3.4 The Pac-Man Domain

In the Pac-Man domain, an agent (Pac-Man) needs to navigate a maze-like environment to collect food pellets (each one gives a +10 reward) while avoiding enemies (ghosts). The G ghosts present in the environment generally move randomly but may start chasing Pac-Man (with probability p_g) if they are close (we conducted our experiments with $p_g = 0.8$). Given the $N \times M$ grid, the agent can move in the four cardinal directions or stay still (hence $|A| = 5$), receiving a time penalty of -1 for each time step spent in the maze. The episode ends with a +500 reward if Pac-Man eats all food pellets or a -500 penalty if one of the ghosts chases Pac-Man. The environment also contains P power capsules (big yellow dots in Fig. 1) that Pac-Man can eat to gain the ability to scare and eat ghosts. Catching a scared ghost results in a +200 reward. The state S contains the position of each wall, food pellet, power pill, ghost in the map, and the Pac-Man position; each is expressed as (x, y) coordinates. The environment is fully deterministic. The challenge Pac-Man presents derives from the vast dimensions of the state space and the extended planning horizon: to complete the level, the agent may have to explore the entire environment to find all the pellets, and non-trivial movements are often required to escape ghosts.

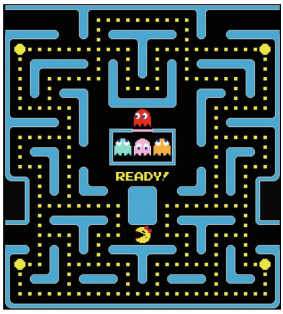


Fig. 1. Example scenario for the Pac-Man domain, $G = P = 4$, 25×26 grid.

4 Methodology

This section describes our methodology for integrating approximate Q-learning, symbolic learning, and reasoning in ASP. We exemplify it in the context of the standard RL Pac-Man domain used for empirical evaluation.

4.1 ASP Representation of the Domain

We start from the representation of the domain in ASP syntax. This requires defining environmental features $\mathcal{F}$ and actions $\mathcal{A}$. For the Pac-Man domain, $\mathcal{F}$ contains: `wall(Dir)`, which denotes the presence of a wall in front of the agent (i.e., one cell away) in that direction, `food(Dir, Dist)`, `ghost(Dir, Dist)`, and `capsule(Dir, Dist)`, representing Manhattan distance `Dist` $\in [0, 10]$ between PacMan and a cell containing food (or ghost, or capsule) in direction `Dir` $\in$ {`north, south, east, west`}. Finally, we need to introduce upper and lower bounds on `Dist` to obtain more informative policy heuristics. To this aim, we define atoms in the form `X_dist_Y(Dir,Dist,D)`, where `X` is either `ghost, food` or `caps`, and `Y` is either `geq` or `leq`, defined as follows:

```
X_dist_geq(Dir, Dist, D) :- X(Dir, Dist), Dist >= D, d_const(D).
X_dist_leq(Dir, Dist, D) :- X(Dir, Dist), Dist <= D, d_const(D).
```

where `d_const(0..4)` limits the set of possible values that `D` can take in the rule. Action atoms are constructed from A as `move(Dir)`, denoting the movement of the agent in direction `Dir`. The agent also has the option to perform a 'stop' action. However, we chose not to include it in the learning phase as it was rarely performed in the examples collected during our tests. Once $\mathcal{F}$ and $\mathcal{A}$ are defined, we need a *feature map* $F_{\mathcal{F}} : \mathrm{S} \to \mathcal{H}(\mathcal{F})$ and an *action map* $F_{\mathcal{A}} : A \to \mathcal{H}(\mathcal{A})$ to ground atoms from collected batches of RL. The only information needed to build $\mathcal{F}$ is the positions of the agent, food, ghosts and capsules in the environment, all of which are available in S.

4.2 Definition of the Learning Task

Given the ASP formalization of the task, we need to generate the ILP task $T = \langle B, S_M, E \rangle$ for FastLAS, starting from RL episodes. Given an episode consisting of a sequence of N state-action pairs $\langle \bar{a}_i, \bar{s}_i \rangle,\ i = 1, \ldots N$, we can build a WCDPI of the following form:

$$e_i = \langle id_i, w_i, \langle \{\bar{a}_i\}, a \in F_{\mathcal{A}} \setminus F_{\mathcal{A}}(\bar{a}_i) \rangle, F_{\mathcal{F}}(\bar{s}_i) \rangle,$$

where id_i is a unique identifier and w_i represents the penalty of the WCDPI, set as the reward obtained by the agent in that episode and assigning $w_i = 0$ to those episodes that obtained a negative reward.[1] This gives more relevance to the agent's behaviour, resulting in a higher reward for generating rules that can lead to at least the same performance. We populate e^{inc} with the agent's chosen action and e^{exc} with the grounding for all unobserved actions. For example, considering the Pac-Man scenario depicted in Fig. 1, if at time t the agent moves left within an episode with return 50, we generate the following WCDPI:

$$\begin{aligned} e_t = &\langle id_t, 50, \langle \texttt{move(east)}, \{\texttt{move(west)}, \texttt{move(north)}, \texttt{move(south)}\} \rangle, \\ &\{\texttt{wall(north)}, \texttt{wall(south)}, \texttt{food(east, 1)}, \texttt{food(west, 1)}, \ldots\} \rangle, \end{aligned} \quad (1)$$

We have omitted ground context atoms with a distance higher than 1 for simplicity. The background knowledge of the task only contains the definition of the ASP variables and ranges, and the mode declaration is defined in order only to have rule's heads in the form `move(Dir)`, and bodies containing atoms from $\mathcal{F}$.

4.3 Neurosymbolic Q-Learning

Algorithm 2 shows our Neurosymbolic Q-Learning methodology. At the end of each batch of experience acquired by Approximate Q-Learning, we store at most σ highest-return episodes (Line 22). We then follow the procedure described in Sect. 4.2 to generate WCDPIs and learn policy heuristics H (Line 23). These heuristics, together with the background knowledge B defined in Sect. 4.2 and the grounding of state variables $F_{\mathcal{F}}(s)$, are used to perform ASP reasoning and compute a set of *suggested actions* A_h at each step of the Q-learner (Line 7). Specifically, in the exploration phase of the agent (with probability ϵ), it may choose either an action from A_h or $A \setminus A_h$ (i.e., actions which H does not suggest), with probability ρ and $1 - \rho$ respectively (Line 10). In this way, we preserve the asymptotic optimality of the Q-learning algorithm [5], following a soft bias approach as in existing literature [13]. We empirically choose ρ as the average discounted return over the best-saved episodes E_σ, normalized by the average return of the whole training. In order not to start with $A_h = \emptyset$ until

[1] This is necessary because FastLAS does not handle negative weights. However, as explained in the following section, we only consider the best episodes of each batch; thus, the presence of rewards less than zero is highly improbable.

the first batch of episodes is gathered, we use the very first episode of training to generate preliminary policy heuristics (Line 21), which provides a greedy yet useful hint to the agent, as explained in Sect. 5. We remark that the described methodology can be equivalently applied to any RL algorithm, where random actions are taken in the exploration phase [5].

Algorithm 2 Neurosymbolic Q-Learning

Require: MDP $= \langle S, A, R, T, \gamma \rangle$, background knowledge B, search space S_M
Parameters: Q-learning parameters [5] (including $\epsilon \in [0, 1]$), max best episodes stored σ, max episodes E, batch size S_b

1: Initialize weight vector $\mathbf{w}$, batch $b = \emptyset$, episode $e = \emptyset$, episode count $N_e = 0$, policy heuristics $H = \emptyset$, set of best episodes $E_\sigma = \emptyset$
2: **while** $N_e < E$ **do**
3: Observe initial state s
4: **while** s is not terminal **do**
5: $x \sim [0, 1]$
6: **if** $x < \epsilon$ **then**
7: $A_h \leftarrow$ **ASP**$(B, H, F_{\mathcal{F}}(s))$
8: **if** $A_h \neq \emptyset$ **then**
9: $\rho \leftarrow$ **AvgReturn**(E_σ)
10: $a \sim$ **WeightedProb**$(A, F_{\mathcal{A}}^{-1}(A_h), \rho)$
11: **else**
12: $a \sim A$
13: **end if**
14: **else**
15: $a \leftarrow \arg\max_{a'} Q(\mathbf{w}, s, a')$
16: **end if**
17: e.Append$(\langle s, a \rangle)$
18: $s' \leftarrow T(s, a)$, $r \leftarrow R(s, a, s')$
19: $\mathbf{w} \leftarrow$ **QUpdate**(s, a, r, s', γ)
20: $s \leftarrow s'$
21: **end while**
22: $N_e \leftarrow N_e + 1$
23: b.**Append**(e)
24: **if** $|b| = S_b \vee N_e = 1$ **then**
25: $E_\sigma \leftarrow$ **Update**(b, σ)
26: $H \leftarrow$ **FastLAS**(B, S_M, E_σ) $\{E_\sigma$ converted to ASP syntax via $F_{\mathcal{F}}, F_{\mathcal{A}}\}$
27: **end if**
28: **end while**

5 Empirical Evaluation

Experimental results on the methodology outlined in the previous section are now presented. We tested our approach on two different scenarios of the Pac-Man

domain: a smaller map, with 18×9 grid dimensions, $P = 2$ power capsules and $G = 2$ ghosts, and a way more challenging map, with 25×26 grid dimensions, $G = 4$ ghosts and $P = 4$ power capsules. All experiments were performed on a computer equipped with 5.1GHz, a 13th Gen Intel i5 processor and 64GB RAM. We evaluate 2 different performance measures:

- *training performance*: it measures the RL discounted return and computational efficiency;
- *policy heuristic convergence*: it measures the explainability and robustness of the symbolic component of our methodology, evaluating how the set of policy heuristics reaches a fixed point as RL training progresses.

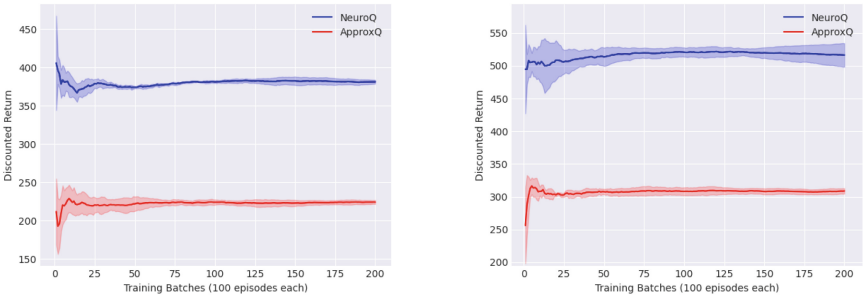
For all metrics, we compare our methodology (**NeuroQ**) against classical Approximate Q-Learning (**ApproxQ**). We report results as mean and standard deviation over a set of 5 random seeds for statistical relevance. For each seed, we choose a maximum number of episodes $E = 20000$ and a batch size $S_b = 100$ (resulting in 200 training batches). Q-Learning parameters for both evaluated algorithms are set as follows: learning rate $\alpha = 0.2, \gamma = 0.8, \epsilon = 0.05$. We empirically choose to keep track of $\sigma = 5$ best episodes on the smaller map, while on the bigger map (in which longer episodes are generated), we set $\sigma = 3$. In this way, we balance RL performance (discounted return) and the computational cost of symbolic learning.

5.1 Training Performance

Figure 2 shows the return obtained by the execution of ApproxQ and NeuroQ in the small map (2a) and in the more challenging map (2b). We note that, in both scenarios, the introduction of policy heuristics significantly increases the return obtained by the agent, leading to a more efficient training process. In addition to assessing the trade-off between performance and computation, we investigated the computational impact of generating policy heuristics and calculating the suggested actions. The execution times, shown in Tables 1, 2 for the small and large maps, respectively, demonstrate that the additional time required by FastLAS learning and ASP reasoning (Lines 7 and 23 in Algorithm 2) is acceptable, given the substantial increase of performance. In particular, in the large map, NeuroQ requires $\approx 25\%$ more time per batch, but the average discounted return is double (Fig. 2b).

5.2 Policy Heuristics Convergence

At each batch iteration of Algorithm 2, the ILP task solved by FastLAS consists of ≈ 650 WCDPIs (on average) for both the small and large maps. The search space S_M remains constant during the procedure and consists of 226 unique rules for both environments. As the first training episode is acquired (Line 21 of Algorithm 2), we learn a first set of policy heuristics to bias the agent's



(a) 18×9 map, $G = P = 2$. (b) 25×26 map, $G = P = 4$.

Fig. 2. Average discounted return in the Pac-Man domain.

Table 1. Execution times (in seconds) in total and per batch, on the 18×9 map with $G = P = 2$. Mean and standard deviation (in brackets) are reported where appropriate.

Seed	ApproxQ		NeuroQ	
	Total	Per batch	Total	Per batch
0	949.39	4.74 (±0.67)	1505.91	7.53 (±0.57)
1	955.86	4.78 (±0.65)	1497.01	7.48 (±0.60)
2	953.72	4.77 (±0.67)	1508.62	7.54 (±0.56)
3	952.75	4.76 (±0.68)	1496.73	7.48 (±0.57)
4	953.46	4.77 (±0.67)	1503.07	7.52 (±0.60)
Average	953.04 (±2.34)	4.77 (±0.67)	1502.27 (±4.92)	7.51 (±0.58)

exploration towards more convenient actions immediately. On the 18×9 map, for example, the following heuristic is generated:

```
move(Dir) :- food_dist_leq(Dir,Dist,1).
```

In other words, the agent immediately learns to move in the direction where the food is close. In the 25×26 map, FastLAS learns a similar heuristic at the first iteration, with slight adjustments depending on the specific seed (e.g., `not wall(Dir)` is included in the body, too).

As the training progresses, learned heuristics finally converge to the following in the smaller environment:

```
move(Dir) :- food_dist_leq(Dir,Dist,1).    (2a)
move(Dir) :- caps_dist_leq(Dir,Dist,1).    (2b)
```

The first greedy heuristic learned in the first episode is confirmed, while the second rule pushes the agent towards close power capsules. On the 25×26 map,

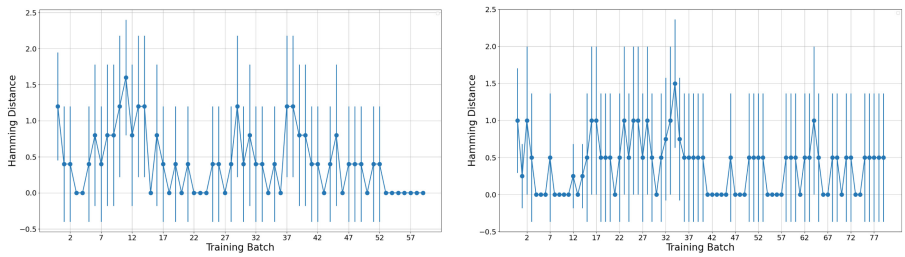
Table 2. Execution times (in seconds) in total and per batch, on the 25×26 map with $G = P = 4$. Mean and standard deviation (in brackets) are reported where appropriate.

Seed	ApproxQ		NeuroQ	
	Total	Per batch	Total	Per batch
0	3980.74	19.9 (±3.26)	4974.46	24.8 (±3.44)
1	4005.92	20.0 (±3.46)	5233.66	26.1 (±3.20)
2	4054.08	20.2 (±3.37)	5197.47	26.0 (±3.16)
3	4003.02	20.0 (±3.45)	5196.34	26.0 (±3.26)
4	4025.39	20.1 (±3.13)	5174.59	25.9 (±3.35)
Average	4013.83 (±27.51)	20.07 (±3.33)	5155.30 (±105.7)	25.78 (±3.28)

the final set of learned rules is the following:

```
move(Dir) :- caps_dist_leq(Dir,Dist,1).                    (3a)
move(Dir) :- food_dist_leq(Dir,Dist,1).                    (3b)
move(Dir) :- not wall(Dir), food_dist_leq(Dir,Dist,2).     (3c)
move(Dir) :- ghost_dist_geq(Dir,Dist,4).                   (3d)
```

In this case, the agent interacts with a more complex map for a longer time. As a consequence, the policy heuristics are more informative and also capture the necessity of avoiding ghosts and walls, stating that the agent should move in a direction only if no walls are present (`not wall(Dir)`) or if ghosts are at a sufficiently high distance (`ghost_dist_geq(Dir,Dist,4)`) in that direction.



(a) 18×9 map, $G = P = 2$ (b) 25×26 map, $G = P = 4$

Fig. 3. Convergence of learned policy heuristics (mean ± standard deviation over seeds). To facilitate visualization, batches beyond convergence are omitted.

Figure 3 shows the convergence of the hypothesis for the small (3a) and large (3b) maps, respectively. Convergence per batch is measured as the Hamming distance between the literals composing the body of rules extracted at a given batch, normalized by the number of literals composing the final sets of rules.

Ideally, we aim for zeroing Hamming distance when convergence is achieved. The charts show that the symbolic learner converges within < 70 batches (over 200) in both scenarios, reflecting the convergence of the overall RL policy. In particular, on the large map (Fig. 3b) rules don't converge to a zeroed Hamming distance (averaging over 5 seeds). Indeed, with one seed, FastLAS alternatively learns two sets of rules, one in which rule 3b is present and one in which it is not. However, rule 3c is always present and is more informative than 3b. Hence, this doesn't represent a significant change in the semantics of the policy.

6 Conclusion and Future Work

The research presented in this paper demonstrates how to effectively integrate neurosymbolic learning and reasoning to improve the efficiency and explainability of RL agents. We leverage the expressive ASP semantics to represent structured task knowledge in a fragment of first-order logic and reason over the most convenient actions in the RL exploration phase. ASP policy heuristics are learned and refined online via a scalable ILP algorithm, gathering examples at each batch of RL training. Moreover, a probabilistic soft bias approach in ASP guidance preserves the convergence guarantees of RL. We validated our algorithm in the Pac-Man benchmark RL scenario in two maps of increasing complexity involving long planning horizons, large state space and contrastive agents (ghosts). Our neurosymbolic RL algorithm extends Q-Learning, but it could be applied to any RL algorithm involving a random exploration phase. Our methodology achieves a significantly higher discounted return than traditional RL (almost double in the largest map), at the cost of $\approx 25\%$ increase in the computational cost. In addition, learned ASP rules converge within $\approx< 70/200$ RL batches, providing an interpretable logical explanation of the black-box RL policy and training process. As a future work, we plan to test the scalability of these techniques to even more complex and varied environments, also investigating the integration with other RL algorithms and more complex forms of neurosymbolic integration [19].

References

1. Vermeulen, A., Manhaeve, R., Marra, G.: An experimental overview of neural-symbolic systems. In: International Conference on Inductive Logic Programming (2023). https://api.semanticscholar.org/CorpusID:266965515
2. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015). https://doi.org/10.1038/nature14236
3. Hasselt, H.V., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: AAAI Conference on Artificial Intelligence (2015). https://api.semanticscholar.org/CorpusID:6208256
4. Mazzi, G., Meli, D., Castellini, A., Farinelli, A.: 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 (2023)

5. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd ed., MIT Press (2018)
6. Meli, D., Nakawala, H., Fiorini, P.: Logic programming for deliberative robotic task planning. Artif. Intell. Rev. **56**(9), 9011–9049 (2023)
7. Marra, G., Kuželka, O.: Neural markov logic networks. In: Uncertainty in Artificial Intelligence, pp. 908–917. PMLR (2021)
8. Furelos-Blanco, D., Law, M., Jonsson, A., Broda, K., Russo, A.: Induction and exploitation of subgoal automata for reinforcement learning. J. Artif. Intell. Res. **70**, 1031–1116 (2021)
9. Meli, D., Sridharan, M., Fiorini, P.: Inductive learning of answer set programs for autonomous surgical task planning. Mach. Learn. **110**(7), 1739–1763 (2021). https://doi.org/10.1007/s10994-021-06013-7
10. Acharya, K., Raza, W., Dourado, C., Velasquez, A., Song, H.H.: Neurosymbolic reinforcement learning and planning: a survey. IEEE Transactions on Artificial Intelligence (2023)
11. Hocquette, C., Muggleton, S.H.: Complete bottom-up predicate invention in metainterpretive learning. In: Proceedings of the Twenty-Ninth International Joint Conferences on Artificial Intelligence, pp. 2312–2318 (2021)
12. Cropper, A., Morel, R.: Learning programs by learning from failures. Mach. Learn. **110**(4), 801–856 (2021). https://doi.org/10.1007/s10994-020-05934-z
13. Meli, D., Castellini, A., Farinelli, A.: Learning logic specifications for policy guidance in POMDPs: an inductive logic programming approach. J. Artif. Intell. Res. **79**, 725–776 (2024)
14. Muggleton, S.: Inductive logic programming. N. Gener. Comput. **8**(4), 295–318 (1991)
15. Lifschitz, V.: Answer set planning. In: International Conference on Logic Programming and Nonmonotonic Reasoning, pp. 373–374. Springer (1999)
16. Law, M., Russo, A., Bertino, E., Broda, K.: FastLAS: scalable inductive logic programming incorporating domain-specific optimisation criteria. Proc. AAAI Conf. Artif. Intell. **34**(3), 2877–2885 (2020). https://doi.org/10.1609/aaai.v34i03.5678
17. Law, M., Russo, A., Broda, K., Bertino, E.: Scalable non-observational predicate learning in ASP. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21) (2021)
18. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP (1988). https://api.semanticscholar.org/CorpusID:261517573
19. Cheng, C.-A., Kolobov, A., Swaminathan, A.: Heuristic-guided reinforcement learning. In: Advances in Neural Information Processing Systems, vol. 34, pp. 13550–13563 (2021)
20. Kambhampati, S., Sreedharan, S., Verma, M., Zha, Y., Guan, L.: Symbols as a lingua franca for bridging human-AI chasm for explainable and advisable AI systems. Proc. AAAI Conf. Artif. Intell. **36**, 12262–12267 (2022)
21. Sridharan, M., Gelfond, M., Zhang, S., Wyatt, J.: REBA: a refinement-based architecture for knowledge representation and reasoning in robotics. J. Artif. Intell. Res. **65**, 87–180 (2019)
22. De Giacomo, G., Iocchi, L., Favorito, M., Patrizi, F.: Foundations for restraining bolts: reinforcement learning with LTLf/LDLf restraining specifications. Proc. Int. Conf. Autom. Plann. Sched. **29**, 128–136 (2019)
23. Leonetti, M., Iocchi, L., Patrizi, F.: Automatic generation and learning of finite-state controllers. In: International Conference on Artificial Intelligence: Methodology, Systems, and Applications, pp. 135–144 (2012)

24. Marzari, L., Marchesini, E., Farinelli, A.: Online safety property collection and refinement for safe deep reinforcement learning in mapless navigation. In: 2023 IEEE International Conference on Robotics and Automation (ICRA), pp. 7133–7139 (2023). https://doi.org/10.1109/ICRA48891.2023.10161312
25. Mazzi, G., Castellini, A., Farinelli, A.: Risk-aware shielding of partially observable monte Carlo planning policies. Artif. Intell. **324**, 103987 (2023)
26. De Giacomo, G., Favorito, M., Iocchi, L., Patrizi, F.: Imitation learning over heterogeneous agents with restraining bolts. Proc. Int. Conf. Autom. Plann. Sched. **30**, 517–521 (2020)
27. Leonetti, M., Iocchi, L., Stone, P.: A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. Artif. Intell. **241**, 103–130 (2016)
28. Hazra, R., De Raedt, L.: Deep explainable relational reinforcement learning: a neuro-symbolic approach. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 213–229. Springer (2023)
29. Veronese, C., Meli, D., Bistaffa, F., Rodríguez-Sot, M., Farinelli, A., Rodríguez-Aguilar, J.A.: Inductive Logic Programming for transparent alignment with multiple moral values. CEUR Workshop Proc. **3615**, 84–88 (2023)
30. Rabold, J., Siebers, M., Schmid, U.: 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 (2018)
31. Rodriguez, I.D., Bonet, B., Romero, J., Geffner, H.: Learning first-order representations for planning from black box states: new results. Proc. Int. Conf. Principles Knowl. Represent. Reasoning **18**, 539–548 (2021)
32. Law, M., Russo, A., Broda, K.: Inductive learning of answer set programs from noisy examples. In: Advances in Cognitive Systems (2018)
33. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, vol. 13, pp. 245–286. Springer (1995)