Heuristic-Guided Distributional Reinforcement Learning

Simone Murari
University of Verona
Verona, Italy
simone.murari@studenti.univr.it

Celeste Veronese
University of Verona
Verona, Italy
celeste.veronese@univr.it

Alessandro Farinelli
University of Verona
Verona, Italy
alessandro.farinelli@univr.it

Daniele Meli
University of Verona
Verona, Italy
daniele.meli@univr.it

ABSTRACT

Distributional Reinforcement Learning (DiRL) is a framework which accounts for the full probability distribution of the action values, resulting in higher training stability and robustness. In standard RL, the agent often requires the aid of domain-specific heuristics to handle long planning horizons and sparse rewards. However, a study on the impact and the best algorithmic integration of heuristics in DiRL is still missing.

This paper addresses this issue, by proposing two different methodologies to handle heuristics at the distributional level in DiRL: modifying the parameters of the distribution (*shift-DiRL*) or altering the full probability mass function (*product-DiRL*). Specifically, we found our research on the most popular C51 algorithm for discrete-action domains, and then we seek to extend our findings to DiRL in continuous action spaces. An empirical analysis over two discrete domains and one continuous shows **the advantage of our methodologies with respect to classical reward machines, even in the case of possibly incorrect heuristics**. Moreover, the superiority of product-DiRL altering the shape of the value probability highlights **the promising role of the distributional representation for heuristic-guided RL**.

KEYWORDS

Distributional Reinforcement Learning, Heuristics for Reinforcement Learning, Learning Agents

1 INTRODUCTION

Reinforcement Learning (RL) selects the optimal action based on the expectation of its value. However, as shown in [1], this often represents a limitation. Indeed, the expectation only encodes one part of information about the environment and the policy, in an inherently stochastic training setting. For this reason, [1] introduced a more holistic perspective on RL, which is Distributional RL (DiRL), considering the full probability distribution of the values of actions, at the stage of policy selection. This brings important advantages, such as enhanced robustness and stability during the training process, as well as the possibility to account for probabilistic risks [19].

On the other hand, as standard RL, also DiRL may struggle in environments with inherent realistic intricacies, e.g., sparse or bad designed reward maps and long planning horizons. In standard RL, and more generally in planning, these issues are typically mitigated with the aid of heuristics, which incorporate external knowledge into the decision-making process, estimating the value of specific states to suggest them with respect to others. Established approaches for heuristic-guided RL include reward shaping [24] and reward machines [15]. Several researchers have also proposed deeper integration of the heuristics at the algorithmic level, to improve the sampling efficiency and the generalization of the learned policies [6, 23]. However, no methodological research nor empirical analysis has been advanced in the field of DiRL so far, practically limiting its application in realistic settings with the aforementioned intricacies.

This paper aims to bridge this gap, by proposing an algorithmic approach to heuristic-guided DiRL. More specifically, we stem from the most popular DiRL algorithm, C51 [1], which is the distributional evolution of the classical Deep Q-Network in discrete action spaces. We then propose two methodologies to integrate heuristics directly in the distributional dimension: either modifying the moments of the value distribution (*shift-DiRL*, inspired from [6]), or altering the full probability mass function (*product-DiRL)*. For both methodologies, we discuss the theoretical aspect and provide a detailed empirical comparison (including ablation studies) in two paradigmatic minigrid scenarios, affected by long planning horizon, sparse reward and partial observability. Then, we propose an additional methodology for heuristic-guided DiRL in continuous action spaces, inspired by product-DiRL and applied to the distributional version of Soft Actor Critic [11]. We also show superior performance with respect to reward machines.

In summary, the contributions of this work are the following;

- We address the problem of heuristic-guided distributional RL, introducing **two methodologies for discrete-action DiRL and one for continuous-action DiRL**;
- We support our methodological contribution with a theoretical discussion on the impact of heuristics on the convergence, and ablation studies for the main parameters of our methods;
- We show the improved performance of our methodologies, also with respect to reward machines [15], in 3 benchmark domains (2 with discrete actions and 1 with continuous actions), characterized by **sparse (potentially non-Markovian) rewards, long planning horizon and possibly incorrect heuristics**. In particular, we find that altering the full return probability mass function with product-DiRL (instead of manipulating moments with shift-DiRL) yields the best

Proc. of the Adaptive and Learning Agents Workshop (ALA 2026), Aydeniz, Delgrange, Mohammedalamen, Yang (eds.), May 25 – 26, 2026, Paphos, Cyprus, https://alaworkshop2026.github.io/. 2026.

performance. **This highlights the importance of integrating heuristics at the distributional level of RL**. Moreover, crucially **our methods do not require RL hyperparameter re-tuning** when the specific environment instance or dimension changes.

2 RELATED WORKS

2.1 Distributional RL

Distributional RL derives from the distributional perspective on Markov Decision Processes (MDPs) and Bellman operator [16, 29]. Having foundations in the human psychological approach to decision-making [20], DiRL exhibits better exploration capabilities than classical expectation-based RL [22], yielding higher robustness and stability during training. Moreover, backed with theoretical foundations [17], it has been mainly employed in the field of risk-sensitive policy learning, both in offline [21] and online RL [7, 28], with application also to challenging robotic tasks [27].

More recently, authors of [1] showed that the benefits of DiRL may go well beyond the quantification of safety. C51, the distributional version of the popular Deep Q-Network for discrete-action MDPs, was then introduced [1, 8], showing improved performance on several benchmark domains with respect to the classical RL counterpart. More recently, distributional algorithms for continuous-action MDPs have also been proposed, including DSAC, the distributional extension to Soft Actor Critic [10, 11].

Beyond the specific application, most research in DiRL has mainly focused on improving the algorithmic efficiency [8] and the informativeness of the value distribution [26], or improving the theoretical understanding of the learning process [17, 25].] Conversely, we focus on the crucial problem of heuristic-guided DiRL, to enhance sample-efficiency, scalability, and ultimately generalization.

2.2 Heuristics for RL

Heuristics are a fundamental pillar of planning and decision-making [3], yielding an estimate of the goodness of specific situations (states), potentially resulting in more efficient action selection and goal convergence. Heuristics are particularly important in RL [2, 4, 5], where they can be extremely helpful to improve sample efficiency and mitigate the effect of sparse or uninformative rewards. For this reason, the most established approach to heuristic-guided RL is reward shaping [24], or more generally reward machines [15], which inject additional information in the reward map to provide more insights about the environment to the agent. The extra reward may be computed based on highly expressive logical specifications [9, 12], in order to mitigate the sparsity due to long planning horizon and deal with sub-goals.

Recently, authors of [6] argued on the inefficiency of heuristic-based reward modifications, suggesting that this does not fully exploit the potential of good informative heuristics, in terms of more efficient training and generalizable policies. For this reason, the most recent works have focused on novel deeper algorithmic integrations of heuristics in RL, guiding the agent especially in the exploration phase [23, 31], which is notably the most sample-inefficient one [18].

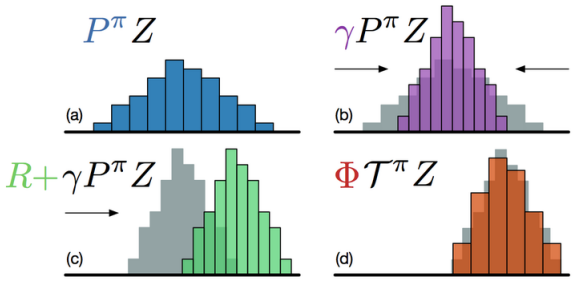


Figure 1: From paper [1]. A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π, (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step

In this paper, we argue that heuristics can be embedded directly in the distributional dimension of DiRL, yielding much better performance in terms of training stability, convergence rate and scalability to complex environments with very large planning horizon and sparse rewards. To the best of our knowledge, this is a novel research direction for the DiRL community.

3 BACKGROUND

We consider the classical setting of a MDP defined as a tuple $\langle S, A, T, R, \gamma \rangle$, where S is the set of states, A is the set of actions, $T : S \times A \times S \to [0, 1]$ is the transition probability that action a in state s transitions to state s', $R : S \times A \to \mathbb{R}$ is the reward map for state-action pairs, and $\gamma \in [0, 1]$ is the discount factor. The objective of RL algorithms is to compute an optimal policy $\pi : S \to A$ for a given MDP, which outputs the probability of selecting an action in a given state. In classical (not distributional) RL, the optimal policy maximizes the Q-value:

$$Q(s, a) = \mathbb{E}Z(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s, a)\right]$$

where $a \sim \pi$. This is practically translated to optimizing Bellman equation:

$$Q(s, a) = \mathbb{E}[R(s, a) + \gamma Q(s', a')] \tag{1}$$

where s', a' denote the next state and action, respectively.

DiRL operates instead directly on $Z(s, a)$ (i.e., the full probability distribution, rather than its expectation). Equation 1 is then reformulated as:

$$Z(s, a) = R(s, a) + \gamma Z(s', a') \tag{2}$$

where $Z(s', a')$ is given by the distributional transitional operator $\mathcal{P}^\pi$, depending on the specific policy π:

$$Z(s', a') = \mathcal{P}^\pi Z(s, a)$$

In the following, we briefly present two main algorithms for DiRL, which represent the foundation for our research: C51 [1] for discrete-action MDPs and DSAC [11] for continuous-action MDPs

3.1 C51

C51 is the distributional extension of Deep Q-Network [30]. C51 represents the value distribution $Z(s,a)$ by a fixed set of N[1] equally spaced supports, called *atoms*, spanning an interval $[V_{\min}, V_{\max}]$ where $V_{\min}, V_{\max} \in \mathbb{R}$. The distribution is then parametrized by a probability mass function over these atoms:

$$Z_\theta(s,a) = z_i \quad w.p. \quad p_i(s,a) = \frac{e^{\theta_i(s,a)}}{\sum_j e^{\theta_j(s,a)}}$$

where the set of atoms is $\{z_i = V_{\min} + i\Delta z : 0 \le i < N\}_{i=1}^{N}$, $\Delta z = \frac{V_{\max} - V_{\min}}{N-1}$ and $\{p_i(s,a)\}$ are their probabilities.

The learning process in C51 involves updating the predicted distribution for a state-action pair (s,a) to match a target distribution constructed from a transition (s,a,r,s'). The process of creating the target distribution is illustrated in Figure 1, and it consists of three steps for a given transition: *discounting* each atom z_j in the support of the next state distribution by the factor γ (resulting in distributional shrinkage); *instantaneous reward summation* according to Equation (2); *re-projection* to align with the fixed support of the predicted distribution.

3.2 DSAC

DSAC is the distributional extension of Soft Actor Critic [13]. The original algorithm was proposed by [10]. It generalizes soft Bellman operator to operate over return distributions. The value distribution $Z(s,a)$ is typically modeled as a Gaussian with mean $Q_\theta(s,a)$ and variance $\sigma^2_\theta(s,a)$. The critic is trained by minimizing the negative log-likelihood of a sampled target return y_z under this distribution:

$$J_Z(\theta) = -\mathbb{E}\left[\log P(y_z \mid Z_\theta(\cdot|s,a))\right]$$

In this paper, we consider the slightly modified version DSAC-T proposed by [11], which improves the training stability and the robustness to reward scale.

4 METHODOLOGY

Before describing our algorithms for heuristic-guided DiRL, we introduce the definition of *action heuristic*, which is used in place of the classical definition of state-based heuristics in decision-making [3].

Definition 1. *An action heuristic $h(s,a)$ is a function which approximates the optimal $Q(s,a)$, as opposed to standard state heuristics $h(s)$ approximating $V(s)$, i.e., the value of a given state.*

While heuristics and action heuristics can be related via the transition map, it is more convenient to use action heuristics in our work, given that our methodology acts on $Z(s,a)$ in DiRL. More specifically, as clarified in the following, we are interested in action heuristics in the form:

$$h(s,a) = \mathbb{1}(s \to a)$$

meaning that $h(s,a) = 1$ iff action a is good in state s. In other words, we are not interested in quantifying the accuracy of the heuristic (i.e., actually approximating the optimal $Q(s,a)$). This approach was already adopted by [23], and requires less parameter tuning than reward machines [12], where the weight of the additional contribution in the reward map must be carefully weighted on a scenario-specific basis.

We now introduce the heuristic-guided adaptations of C51 (H-C51) and DSAC (H-DSAC-T)[2].

4.1 H-C51 - Heuristic Guided C-51

Algorithm 1 Heuristic-Guided C51 Training Loop

Require: Q-network Z_θ, target network $Z_{\bar\theta}$, Replay buffer $\mathcal{D}$, rule distribution ρ, atom support $\{z_i\}_{i=0}^{N-1}$, $\epsilon_i = 1, \epsilon_r, \epsilon_f$

1: **Initialize:** Q-network Z_θ, target network $Z_{\bar\theta} \leftarrow Z_\theta$
2: **for** each training step $t = 1, \dots, T$ **do**
3: Observe state s_t
4: $\epsilon_t = max(\epsilon_i - t \cdot \epsilon_r, \epsilon_f)$
5: $x \sim \mathcal{U}(0,1)$
6: **if** $x < \epsilon_t$ **then**
7: Select $a_t \sim$ H-Exploration(s_t)
8: **else**
9: Select $a_t \sim$ H-Exploitation(s_t, ϵ_t) ▷ Shift
or
10: Select $a_t \sim$ H-Exploitation(s_t, ϵ_t, ρ) ▷ Product
11: Execute a_t, observe r_t, s_{t+1} and store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
12: Sample mini-batch of B transitions (s,a,r,s') from $\mathcal{D}$
13: **for** each transition (s,a,r,s') in the batch **do**
14: **for** each $a' \in \mathcal{A}$ **do**
15: Predict $p_i(s',a')$
16: Compute $Q(s',a') = \sum_i^{N-1} z_i p_i(s',a')$
17: $a^* \leftarrow \arg\max_{a'} Q(s',a')$
18: $Tz_j \leftarrow r + \gamma z_j$
19: Project Tz_j onto support $\{z_i\}$ via m_i:
20: **for** $j = 0$ to $N-1$ **do**
21: $b_j \leftarrow (Tz_j - V_{\min})/\Delta z$
22: $l \leftarrow \lfloor b_j \rfloor, \quad u \leftarrow \lceil b_j \rceil$
23: $m_l \leftarrow m_l + p_j(s',a^*) \cdot (u - b_j)$
24: $m_u \leftarrow m_u + p_j(s',a^*) \cdot (b_j - l)$
25: Update θ by minimizing cross-entropy:

$$\mathcal{L} = -\sum_i m_i \log p_i(s,a)$$

26: Periodically update target network $\bar\theta$

The training loop of H-C51 is shown in Algorithm 1. The algorithm proceeds analogously to the original C51, balancing exploration vs. exploitation according to a modulated exploration factor ϵ_t, which decreases linearly (via ϵ_r) in $[\epsilon_i, \epsilon_f]$. Action heuristics are integrated in the lines highlighted in red. Specifically, at Line 7 we have heuristic-guided exploration, outlined in Algorithm 2, which selects actions based on a heuristic-induced probability distribution (see Line 2 of Algorithm 2), instead of the classical uniform distribution[3]. This strategy is inspired from related works in

[1] $N = 51$ in [1], thus justifying the name of the algorithm.

[2] In the following algorithms, where we do not alter the original methodologies, we report the standard notation used in the original papers of C51 [1] and DSAC [11]; the reader is referred to them for symbol understanding.

[3] $\rho_{\max}$ is set empirically to 0.8 in our experiments. It does not significantly impact the training performance, but it must hold $\rho_{\max} < 1$ to avoid action shielding, hence incomplete exploration.

heuristic-guided Monte Carlo tree search [23], and is implemented to address the challenges of RL exploration in sparse reward settings [18].

Algorithm 2 H-Exploration(s)

1: **for** each action $a \in \mathcal{A}$ **do**
2: $\quad w_a \leftarrow \begin{cases} \rho_{\max} & \text{if } h(s,a) = 1 \\ 1 - \rho_{\max} & \text{otherwise} \end{cases}$
3: Normalize weights: $\tilde{w}_a \leftarrow \frac{w_a}{\sum_{a'} w_{a'}}$
4: Sample $a \sim \textsc{WeightedSample}(\tilde{w})$
5: **return** a

In the exploitation phase, we propose two alternative algorithmic integrations of action heuristics at the distributional level (Lines 9-10 of Algorithm 1). With the *shift-DiRL* strategy (Line 9) reported in Algorithm 3, we penalize not suggested actions by shifting their predicted return distributions toward lower values. This acts similarly as step (c) in Figure 1. As shown in Algorithm 3, the probability mass of each non-suggested action is shifted to the left by $k = \lfloor \epsilon \cdot N \rfloor$ atoms[4] (lines 3 to 5), where ϵ is the current exploration rate and N is the number of atoms. The $\lfloor \cdot \rfloor$ operator is used to ensure that the shift amount is a valid integer, since the number of atoms is fixed. The shift is implemented by discarding the first k probability values (corresponding to the lowest-return atoms) and appending k zeros to the high-return end of the distribution. This modification reduces the expected return of non-suggested actions without affecting the distributions of those that are heuristically suggested.

Algorithm 3 H-Exploitation Shift(s, ϵ)

1: **for** each action $a \in \mathcal{A}$ **do**
2: $\quad$ Predict $p(s,a)$
3: $\quad$ **if** $h(s,a) = 0$ **then**
4: $\quad\quad k \leftarrow \lfloor \epsilon \cdot N \rfloor$
5: $\quad\quad p(s,a) \leftarrow \textsc{LeftShift}(p(s,a), k)$
6: Normalize: $\tilde{p}(s,a) \leftarrow \frac{p(s,a)}{\sum_a' p(s,a')}$
7: **for** each action $a \in \mathcal{A}$ **do**
8: $\quad$ Compute $Q(s,a) \leftarrow \sum_i^{N-1} z_i \tilde{p}_i(s,a)$
9: $a^* \leftarrow \arg\max_a Q(s,a)$
10: **return** a^*

On the other hand, Line 10 of Algorithm 1 reports an alternative *product-DiRL* strategy, which is outlined in Algorithm 4 and does not act only on the mean, but rather on the full value distribution. More specifically, this algorithm reweights the predicted distribution $p(s,a)$ for each action a suggested by the heuristic, which is multiplied element-wise by a fixed sigmoidal distribution ρ scaled by the current exploration factor ϵ (line 4). The distribution ρ is defined once at initialization and is centered at high (normalized) return values[5] to emphasize atoms near the maximum return, thereby biasing the expected return estimate toward high values.

Algorithm 4 H-Exploitation Product(s, ϵ, ρ)

1: **for** each action $a \in \mathcal{A}$ **do**
2: $\quad$ Predict $p(s,a)$
3: $\quad$ **if** $h(s,a) = 1$ **then**
4: $\quad\quad p(s,a) \leftarrow p(s,a) \cdot (1 + \epsilon \cdot \rho)$
5: Normalize: $\tilde{p}(s,a) \leftarrow \frac{p(s,a)}{\sum_a' p(s,a')}$
6: **for** each action $a \in \mathcal{A}$ **do**
7: $\quad$ Compute $Q(s,a) \leftarrow \sum_i^{N-1} z_i \tilde{p}_i(s,a)$
8: $a^* \leftarrow \arg\max_a Q(s,a)$
9: **return** a^*

4.2 H-DSAC-T - Heuristic-guidede DSAC-T

Algorithm 5 shows the heuristic-guided distributional variant of DSAC-T, with our modifications in red. The DSAC-T pipeline from [11] is implemented, which modifies the actor component of the DSAC-T framework by modifying the mean of the stochastic policy. Specifically, the policy $\pi(a|s)$ is modeled as a Gaussian distribution with subsequent tanh normalization, with a state-dependent mean $\mu(s)$ and standard deviation $\sigma(s)$. At Line 7, Algorithm 5 replaces the original mean with an interpolated vector $\mu_{\text{guided}}(s)$, computed as a linear combination of the learned mean and a suggested action a_h such that $h(s,a_h) = 1$:

$$\mu_{\text{guided}}(s) = (1 - \epsilon_t) \cdot \mu(s) + \epsilon_t \cdot a_h$$

This modification biases the center of the distribution toward the heuristic suggestion before the application of the tanh squashing function, influencing the sampled action with a modulation depending on the exploration factor. Given the tanh re-modulation, this algorithm has a similar effect to HC51-Product[6], altering the final shape of the distribution.

4.3 Theoretical analysis

The heuristic-guided algorithms introduced in this section are designed to preserve the convergence properties of their respective base algorithms (C51 and DSAC-T). This is achieved by ensuring that the integration of heuristic guidance affects only the action selection mechanism, without altering the learning updates of the respective algorithms.

In this regard, we observe that the heuristics do not restrict the action space through shielding. Instead, they modify the sampling distribution over actions by assigning higher probabilities to those recommended by the heuristic or penalizing those not recommended. As a result, the optimal policy remains reachable even in the presence of heuristic guidance, provided that training is allowed to proceed for a sufficiently long time.

[4] An alternative design could shift the PMFs of the *suggested* actions to the right (i.e., toward higher-return atoms). While this approach may appear more intuitive, empirical results showed that penalizing non-suggested actions leads to more stable training in our setting.

[5] Empirically, we center ρ at 0.8 in our experiments. In the experiments, we show the impact of varying ε; **since ρ is multiplied by ε, this has the same effect as studying the impact of ρ.**

[6] This will be evidenced as the best methodology for discrete-action MDPs in the experiments.

Algorithm 5 H-DSAC-T

Require: $\theta_1, \theta_2, \phi, \alpha, \beta_Z, \beta_\pi, \beta_\alpha, \tau$, heuristic action $a_h, \epsilon_i = 1, \epsilon_r, \epsilon_f$

1: Initialize target networks: $\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2, \bar{\phi} \leftarrow \phi$
2: **for** each training step $t = 1, \ldots, T$ **do**
3: **for** each sampling step **do**
4: Observe state s
5: Compute actor mean $\mu(s)$
6: $\epsilon_t = max(\epsilon_i - t \cdot \epsilon_r, \epsilon_f)$
7: $\mu_{\text{guided}}(s) \leftarrow (1 - \epsilon_t) \cdot \mu(s) + \epsilon_t \cdot a_h$
8: Sample action $a \sim \mathcal{N}(\mu_{\text{guided}}(s), \sigma(s))$
9: Execute a, observe reward r and next state s'
10: Store (s, a, r, s') in buffer $\mathcal{B}$
11: **for** each update step n **do**
12: Sample $s, a, r, s' \sim \mathcal{B}$
13: Compute actor mean $\mu(s)$
14: $\epsilon_n = \epsilon_t$
15: $\mu_{\text{guided}}(s) \leftarrow (1 - \epsilon_n) \cdot \mu(s) + \epsilon_n \cdot a_h$
16: Sample action $a \sim \mathcal{N}(\mu_{\text{guided}}(s), \sigma(s))$
17: Update critic using $\theta \leftarrow \theta - \beta_Z \nabla_\theta J_Z^{\text{scaled}}(\theta)$
18: Update actor using $\phi \leftarrow \phi + \beta_\pi \nabla_\phi J_\pi(\phi)$
19: Update temperature:
$$\alpha \leftarrow \alpha - \beta_\alpha \mathbb{E}_{s\sim\mathcal{B}, a\sim\pi_\phi}\left[-\log \pi_\phi(a|s) - \mathcal{H}\right]$$
20: Update target networks:
$$\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}, \quad \bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}$$

Moreover, the algorithmic integrations are modulated via the exploration factor ϵ_t, which decays as the training progresses (Line 4 of Algorithm 1 and Line 6 of Algorithm 5). On the one hand, this schedule ensures that the heuristic has a stronger impact during early training, hence significantly supporting the agent in the early exploration stages. On the other hand, the decaying rule reduces the impact of badly designed heuristics in the long term, thus not severely affecting the convergence rate of the original distributional algorithms in the worst-case scenarios.

Finally, we remark that our algorithms do not affect the target distributions, thus preserving the original theoretical properties.

5 EXPERIMENTS

We validate our methodology in three benchmark tasks (Figure 2). In the discrete-action domain, we consider *Door Key* from MiniGrid environments[7] and *Office World* [15]. For continuous actions, we consider the *Reacher-v2* task[8] for robotic manipulation. **All tasks involve sparse rewards and potentially long planning horizons, thus necessitating the distributional robustness and heuristics.**

We compare our algorithms against the baseline distributional algorithms and the state-of-the-art reward machine (*RM*) approach [15], where an additional reward is given to the agent when it takes an action a at state s such that $h(s, a) = 1$. **Importantly, when the specific instance of the environment is changed (e.g., larger or richer grids), we do not re-tune RL hyper-parameters[9].** In addition, **the heuristics for our domains are imperfect (possibly incorrect)**, in that they are learned either symbolically by [14] (Door Key) and [12] (Office), or via RL for Reacher (see the following sections for details). In this way, we aim at highlighting **the generalization of our methodologies to complex tasks, where correct heuristic definition is hard**.

Finally, we perform an ablation study to assess the impact of two crucial parameters of our methodologies: the ϵ-decay profile and the role of action heuristics in exploration vs. exploitation[10]

5.1 Door Key

This is a partially observable grid-based navigation environment (Figure 2a) in which the agent (red arrow) must pick up a key, open a colour-matching door, and reach the goal location (green).

The observation space for the MDP corresponds to a 7×7 area (light gray), with each cell encoded as object type, colour and state. The discrete action space contains five actions: `turn left`, `turn right`, `move forward`, `pick up` and `toggle` (used to open the door).

The agent receives a reward only upon reaching the goal after opening the required door. If the goal is reached, the reward is computed as $r(s, a) = 1 - 0.9 \cdot \frac{\text{step\_count}}{\text{max\_steps}}$, where max_steps = env_size · env_size · 10 is the default value defined by the environment. Otherwise, the reward is zero.

The action heuristics are expressed as the following logical rules as in [14]:

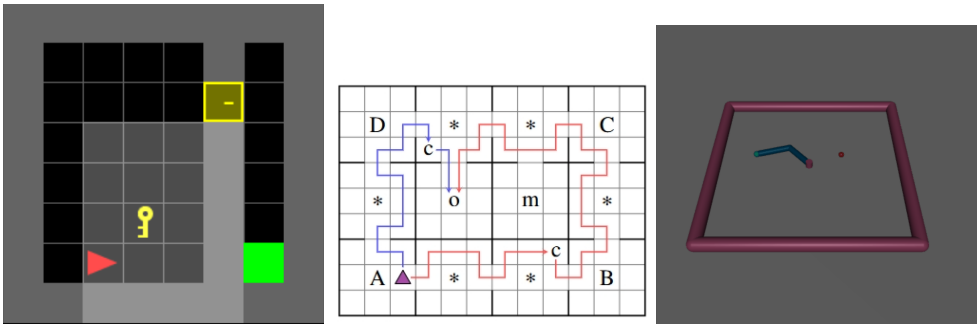
(1) `pickup(X) :- key(X), sameColour(X, Y), door(Y), notCarrying`
(2) `toggle(X) :- door(X), locked(X), carryingKey(Z), sameColour(X,Z)`
(3) `goto(X) :- goal(X), unlocked`

Rule 1 suggests that the agent should pick up a key X if it matches the colour of a door Y and the agent is not currently carrying another key. Rule 2 states that the agent can unlock a door X if it is carrying a key Z of the same colour and the door is currently locked. Finally, Rule 3 instructs the agent to move towards the goal X if the path is clear and the door is already unlocked. Unlike the first two rules, this one does not correspond directly to a specific action in the MDP; instead, it is interpreted as a high-level directive to navigate towards the goal.

5.1.1 Results. Figure 3 shows the results of our methodologies H-C51 Product (blue) and Shift (pink) against the baselines in different maps. RL hyper-parameters were tuned on the smallest map (8×8 with 1 key). Our algorithms perform similarly one to each other, and always outperform in terms of convergence rate, and often in terms of final convergence value of the return. Also, the training curves are generally more stable with respect to the reward machine approach, in line with the known limitation of reward modifications [6]. Notably, *RM* (red) is not able to outperform even the baseline C51 in semantically rich settings with 4 keys (Figures 3c-3f), where the role of heuristics is crucial to pick the key with the correct colour.

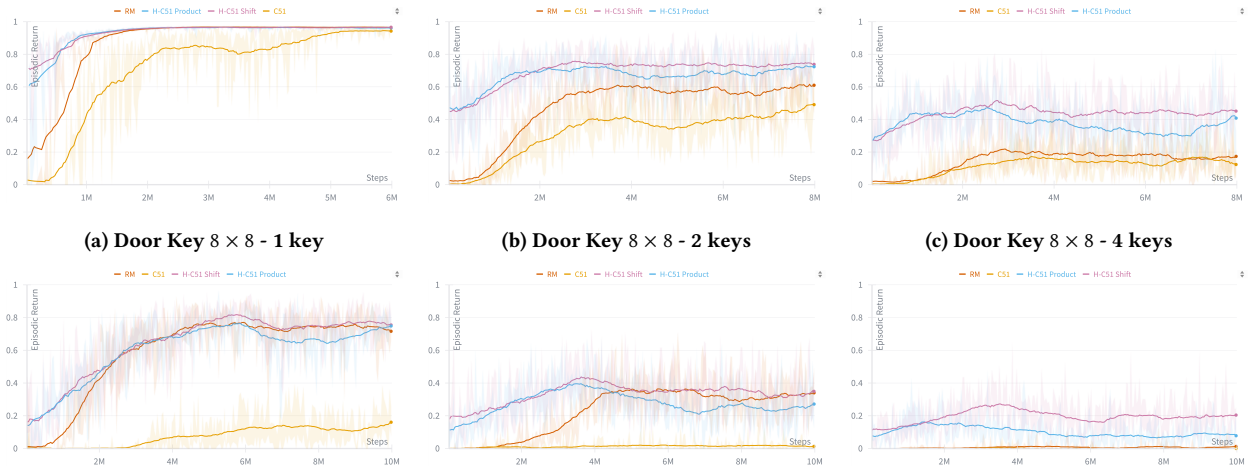
[7] https://minigrid.farama.org/environments/minigrid/DoorKeyEnv/
[8] https://gymnasium.farama.org/environments/mujoco/reacher/

[9] We only tune the episode length and the batch size as the planning horizon increases.
[10] The latter is performed only for H-C51, since in H-DSAC-T there is no neat separation between exploration and exploitation.



(a) Door Key 8×8 **with one key** **(b) Office** **(c) Reacher**

Figure 2: Benchmark domains.



(a) Door Key 8×8 **- 1 key** **(b) Door Key** 8×8 **- 2 keys** **(c) Door Key** 8×8 **- 4 keys**

(d) Door Key 16×16 **- 1 key** **(e) Door Key** 16×16 **- 2 keys** **(f) Door Key** 16×16 **- 4 keys**

Figure 3: Door Key results.

5.2 Office

Office World (Figure 2b) is a grid-based environment, consisting of a rectangular grid populated by static labeled locations and objects such as mail (m), coffee (c), office (o) and decorative plants ($*$), which act as obstacles. Additional rooms labeled A through D are present and have to be accessed depending on the specific task variant. The agent (red arrow) can move deterministically using four actions: `up`, `down`, `left`, `right`.

The observation space for the MDP includes the agent's position on the grid, whether the agent is currently holding the coffee, whether it is holding the mail, and the progress in room visitation. The reward is sparse at goal reaching, $r(s,a) = 1 - 0.9 \cdot \frac{\text{step\_count}}{\text{max\_steps}}$, where max_steps = env_width · env_height · 10 by default. Otherwise, the reward is zero.

We consider two variants of the general task:

- *Deliver coffe and mail*, where the agent must collect both coffee and mail before reaching the office. The following heuristics by [12] are used:

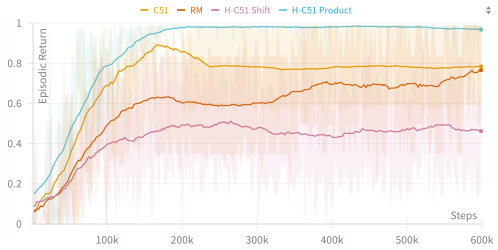
(1) `goto(X) :- coffee(X), not HasCoffee, notHittingPlants.`
(2) `goto(X) :- office(X), HasCoffee, notHittingPlants.`
This heuristic provides information about one sub-task, i.e., delivering coffee.

- *Patrol ABC*, where the agent must visit three rooms labeled A, B and C in the specified order while avoiding plants. ***This is a very sparse-reward with three sequential sub-goals, critically requiring action heuristics.*** The corresponding heuristic by [12] are:

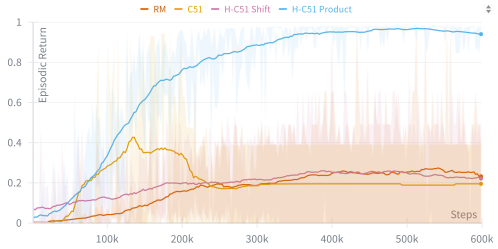
(1) `goto(X) :- room_a(X), not visited_a, not visited_b, not visited_c, notHittingPlants.`

(2) `goto(X) :- room_b(X), visited_a, not visited_b, not visited_c, notHittingPlants.`

(3) `goto(X) :- room_c(X), visited_a, visited_b, not visited_c, notHittingPlants.`

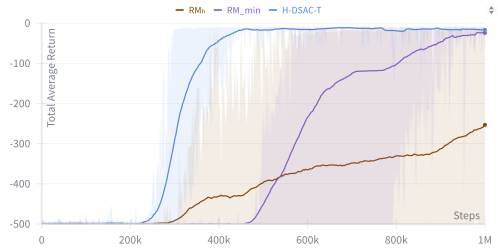


(a) Deliver coffe and mail

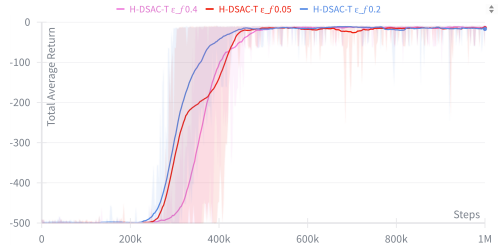


(b) Patrol ABC

Figure 4: Office results.



(a) Main results



(b) ϵ study

Figure 5: Reacher results.

5.2.1 Results. Figure 4 shows the results for the Office domain. Here, H-C51 Product (blue) significantly outperforms the other baselines. In particular, H-C51 Shift (pink) performs similarly to RM (red) and base C51 (yellow) in patroling (Figure 4b), but performs even worse when delivering coffee and mail (Figure 4a). These results possibly depend on the randomization of the initial configuration of the environment at the beginning of each episode, indicating the higher robustness of the Product strategy. H-C51 Product also attains the lowest variance, hence the best training stability. The significantly lower average return achieved by the other baselines in patroling, as well as the lower convergence, are indicators of the harder complexity of the task.

5.3 Reacher

This is a continuous control task in which a two-joint robotic arm must reach a static target (Figure 2c).

The observation space for the MDP consists of 11 continuous features: the sine and cosine of each joint angle, the coordinates of the target, the angular velocities of both joints and the 3-dimensional vector from the target to the fingertip. The task is two-dimensional, so the z-component of this vector is always zero, but it is included in the observation. The action space is continuous and 2-dimensional, representing the torques applied to each joint.

Let $p_f(s), p_t(s) \in \mathbb{R}^2$ denote the fingertip and target positions at state s, respectively, and let $\| \cdot \|$ be the Euclidean norm. The sparse reward is

$$r(s,a) = \begin{cases} 0 & \text{if } \| p_f(s) - p_t(s) \| \le \varepsilon \\ -1 & \text{otherwise} \end{cases} \qquad \text{with } \varepsilon = 0.01$$

Since no state-of-the-art heuristic is known, one is derived empirically by training a DSAC-T agent (RM_{min}) to convergence. In this variant, the reward was modified to minimize the norm of the applied torques, in order to identify the minimal torque necessary to complete the task

$$r(s,a) = \begin{cases} -\|a\|^2 & \text{if } \| p_f(s) - p_t(s) \| \le \varepsilon \\ -1 - \|a\|^2 & \text{otherwise} \end{cases} \qquad \text{with } \varepsilon = 0.01$$

where $a \in \mathbb{R}^2$ is the action vector.

We then extract the average action from episodes at convergence. This vector, denoted $a_h = [0.25, -0.45]$, serves as a continuous proxy for a binary heuristic function $h(s,a)$, by biasing the policy mean in the direction of a low-torque action known to succeed. More specifically, $h(s,a)$ is then defined as:

$$h(s,a) = \begin{cases} 1 & \text{if action } a = a_h \\ 0 & \text{otherwise} \end{cases}$$

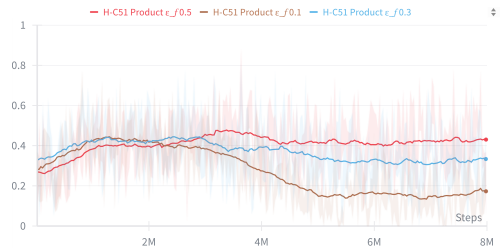
5.3.1 Results. Figure 5 shows the performance of H-DSAC-T (blue) against RM_{min} (minimizing the norm of the torques, purple) and RM approach with a_h used for building the reward machine (brown), i.e., by assigning an additional negative reward proportional to $-\|a - a_h\|^2$. Our method achieves significantly faster and more stable convergence.

5.4 Ablation and parameter studies

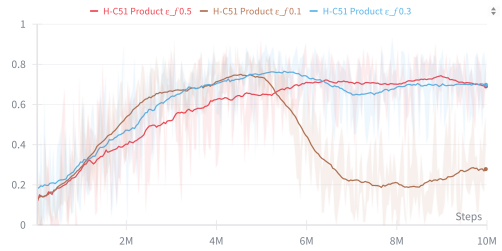
We now analyze the impact of two crucial parameters of our methodologies: the ϵ-decay profile and the role of action heuristics in exploration vs. exploitation[11].

5.4.1 ϵ-decay profile. The ϵ-decay profile is crucial for the performance of all our algorithms, since it modulates the impact of the

[11]The latter is performed only for H-C51, since in H-DSAC-T there is no neat separation between exploration and exploitation.

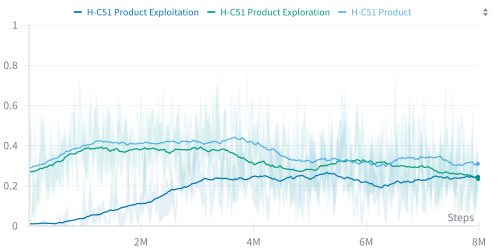


(a) Door Key 8×8**, 4 key**

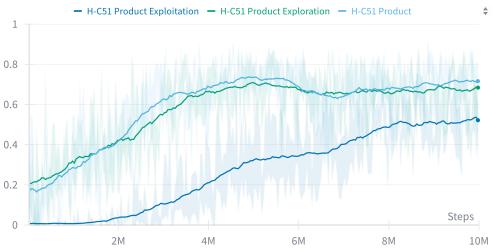


(b) Door Key 16×16**, 1 key**

Figure 6: Impact of ϵ-decay profile.



(a) Door Key 8×8**, 4 key**



(b) Door Key 16×16**, 1 key**

Figure 7: Impact of exploration vs. exploitation.

heuristics on the training loops (Algorithms 1-5). In particular, the decay law is governed by the final value of ϵ_f, which determines the speed of discarding exploration in favor of exploitation[12]. Thus, we now study the impact of this parameter on the performance of H-C51 Product[13] and H-DSAC-T. For H-C51, we report the results for Door Key with 4 keys in an 8×8 map (Figure 6a), and 1 key in a 16×16 map (Figure 6b), in order to separately highlight the impact of long planning horizon (larger grid) and rich environmental semantics and options for the agent (more keys). Figure 6 shows that lower values for ϵ_f (brown curves, with respect to the optimal value of 0.3 used in the previous experiments, blue curves) worsen the final return at convergence, especially when the planning horizon increases (larger map). This confirms the importance of a suitable amount of exploration on RL performance in highly-sparse reward maps [18].

Figure 5b evidences the impact of ϵ_f on H-DSAC-T performance. While the optimal value of previous experiments ($\epsilon_f = 0.2$, blue curve) leads to a faster and more stable growth of the return, the impact of the ϵ-decay trend on DiRL performance appears minor with respect to H-C51, possibly due to the better performance of the base DSAC-T algorithm.

5.4.2 Exploration vs. exploitation. Figure 7 shows the performance of H-C51 Product when either H-Exploration (Algorithm 2) or H-Exploitation (Algorithm 4) is omitted from the full Algorithm 1. From both charts, it is evident that ***H-Exploration has the highest impact on the faster growth of the training curve***, in agreement with the importance of exploration for RL [18]. However, on the 16×16 map (Figure 7b) ***with a long planning horizon, it is also evident the utility of H-Exploitation***, which achieves higher average return (≈ 0.5, blue curve) than base C51 (yellow curve in Figure 3d).

[12] ϵ_r has a similar impact, assuming a linear decay of the exploration factor. Hence, we only study the impact of ϵ_f.

[13] This is chosen as the best-performing algorithm for discrete-action spaces, as shown in the previous experiments.

6 CONCLUSION

This paper explored the problem of effectively integrating heuristics in distributional RL, in order to foster its scalability and application to complex domains characterized by sparse (potentially non-Markovian) rewards, long planning horizons and partial observability or sub-goals. We proposed two algorithms for heuristic-guided C51 (discrete actions) and one algorithm for heuristic-guided DSAC-T (continuous actions). The algorithms do not alter the theoretical convergence properties of the respective baselines. However, in the context of 2 discrete-action settings and 1 continuous control problem, **our solutions allow significantly faster and more stable convergence rate against state-of-the-art reward machines, in scenarios of increasing complexity and with possibly incorrect heuristics. Moreover, altering the full return probability mass (and not just its moments) suggests the promising role of heuristics in a distributional representation of RL.**

We believe this work clearly shows the benefits of using heuristics in distributional RL and it paves the way towards several interesting research lines. These include the evaluation of the proposed methodologies in more complex or risk-sensitive tasks, including real robotic applications in continuous-action settings.

REFERENCES

[1] Marc G Bellemare, Will Dabney, and Rémi Munos. 2017. A distributional perspective on reinforcement learning. In *International conference on machine learning*. PMLR, 449–458.

[2] Reinaldo AC Bianchi, Luiz A Celiberto Jr, Paulo E Santos, Jackson P Matsuura, and Ramon Lopez De Mantaras. 2015. Transferring knowledge as heuristics in

reinforcement learning: A case-based approach. *Artificial Intelligence* 226 (2015), 102–121.

[3] Blai Bonet and Héctor Geffner. 2001. Planning as heuristic search. *Artificial Intelligence* 129, 1-2 (2001), 5–33.

[4] Tim Brys. 2016. Reinforcement Learning with Heuristic Information. *Dissertationm Vrije Universiteit Brussel* (2016).

[5] Luiz A Celiberto Jr, Carlos HC Ribeiro, Anna HR Costa, and Reinaldo AC Bianchi. 2007. Heuristic reinforcement learning applied to robocup simulation agents. In *Robot Soccer World Cup*. Springer, 220–227.

[6] Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. 2021. Heuristic-guided reinforcement learning. *Advances in Neural Information Processing Systems* 34 (2021), 13550–13563.

[7] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. 2018. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*. PMLR, 1096–1105.

[8] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. 2018. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[9] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. 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. 128–136.

[10] Jingliang Duan, Yang Guan, Shengbo Eben Li, Yangang Ren, Qi Sun, and Bo Cheng. 2021. Distributional soft actor-critic: Off-policy reinforcement learning for addressing value estimation errors. *IEEE transactions on neural networks and learning systems* 33, 11 (2021), 6584–6598.

[11] Jingliang Duan, Wenxuan Wang, Liming Xiao, Jiaxin Gao, Shengbo Eben Li, Chang Liu, Ya-Qin Zhang, Bo Cheng, and Keqiang Li. 2025. Distributional Soft Actor-Critic With Three Refinements. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 47, 5 (2025), 3935–3946. https://doi.org/10.1109/TPAMI.2025.3537087

[12] Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda, and Alessandra Russo. 2021. Induction and exploitation of subgoal automata for reinforcement learning. *Journal of Artificial Intelligence Research* 70 (2021), 1031–1116.

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. Pmlr, 1861–1870.

[14] Rishi Hazra and Luc De Raedt. 2023. Deep explainable relational reinforcement learning: a neuro-symbolic approach. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 213–229.

[15] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2022. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research* 73 (2022), 173–208.

[16] Stratton C Jaquette. 1976. A utility criterion for Markov decision processes. *Management Science* 23, 1 (1976), 43–49.

[17] Tyler Kastner, Murat A Erdogdu, and Amir-massoud Farahmand. 2023. Distributional model equivalence for risk-sensitive reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2023), 56531–56552.

[18] Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. 2022. Exploration in deep reinforcement learning: A survey. *Information Fusion* 85 (2022), 1–22.

[19] Shiau Hong Lim and Ilyas Malik. 2022. Distributional reinforcement learning for risk-sensitive policies. *Advances in Neural Information Processing Systems* 35 (2022), 30977–30989.

[20] Adam S Lowet, Qiao Zheng, Sara Matias, Jan Drugowitsch, and Naoshige Uchida. 2020. Distributional reinforcement learning in the brain. *Trends in neurosciences* 43, 12 (2020), 980–997.

[21] Yecheng Ma, Dinesh Jayaraman, and Osbert Bastani. 2021. Conservative offline distributional reinforcement learning. *Advances in neural information processing systems* 34 (2021), 19235–19247.

[22] Borislav Mavrin, Hengshuai Yao, Linglong Kong, Kaiwen Wu, and Yaoliang Yu. 2019. Distributional reinforcement learning for efficient exploration. In *International conference on machine learning*. PMLR, 4424–4434.

[23] Daniele Meli, Alberto Castellini, and Alessandro Farinelli. 2024. Learning logic specifications for policy guidance in pomdps: an inductive logic programming approach. *Journal of Artificial Intelligence Research* 79 (2024), 725–776.

[24] Henrik Müller and Daniel Kudenko. 2025. Improving the Effectiveness of Potential-based Reward Shaping in Reinforcement Learning. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*. 2684–2686.

[25] Mark Rowland, Marc Bellemare, Will Dabney, Rémi Munos, and Yee Whye Teh. 2018. An analysis of categorical distributional reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 29–37.

[26] Mark Rowland, Robert Dadashi, Saurabh Kumar, Rémi Munos, Marc G Bellemare, and Will Dabney. 2019. Statistics and samples in distributional reinforcement learning. In *International Conference on Machine Learning*. PMLR, 5528–5536.

[27] Lukas Schneider, Jonas Frey, Takahiro Miki, and Marco Hutter. 2024. Learning risk-aware quadrupedal locomotion using distributional reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 11451–11458.

[28] Rahul Singh, Qinsheng Zhang, and Yongxin Chen. 2020. Improving robustness via risk averse distributional reinforcement learning. In *Learning for Dynamics and Control*. PMLR, 958–968.

[29] Matthew J Sobel. 1982. The variance of discounted Markov decision processes. *Journal of Applied Probability* 19, 4 (1982), 794–802.

[30] Richard S Sutton, Andrew G Barto, et al. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[31] Celeste Veronese, Daniele Meli, and Alessandro Farinelli. 2024. Online inductive learning from answer sets for efficient reinforcement learning exploration. In *International Workshop on Hybrid Models for Coupling Deductive and Inductive Reasoning*. Springer, 93–106.