Using Incomplete and Incorrect Plans to Shape Reinforcement Learning in Long-Sequence Sparse-Reward Tasks

Henrik Müller
L3S Research Center
Hannover, Germany
hmueller@l3s.de

Lukas Berg
L3S Research Center
Hannover, Germany
lukas.berg@l3s.de

Daniel Kudenko
L3S Research Center
Hannover, Germany
kudenko@l3s.de

ABSTRACT

Reinforcement learning (RL) agents naturally struggle with long-sequence sparse reward tasks due to the lack of reward feedback during exploration and the problem of identifying the necessary action sequences required to reach the goal. Previous works have used abstract symbolic task knowledge models to speed up RL agents in these tasks by either splitting the task into easier to solve subtasks or by creating an artificial dense reward function. These approaches are often limited by their requirement of perfect symbolic knowledge models, which cannot be guaranteed when the abstract symbolic models are provided by humans and in real world tasks. We introduce *Exponential Plan-Based Reward Shaping*, which is able to leverage the ability to learn from experience of RL to compensate deficiencies in incomplete and incorrect abstract symbolic plans and use them to solve difficult tasks faster, while guaranteeing convergence to the optimal policy. Our approach is able to work with plans that miss important steps, include unnecessary extra steps, contain steps that refer ambiguously to both important and useless states, or encode an incorrect order of steps. We use action representations designed by human experts to automatically compute plans to capture the high-level task structure. The abstract symbolic subgoals defined by the plan are used to create dense reward feedback, which signals important states to the RL agent that should be achieved and explored to reach the goal. We show the theoretical advantages of our approach for plans with many steps and show its effectiveness empirically on multiple tasks with different kinds of incomplete or incorrect knowledge.

KEYWORDS

Reinforcement Learning, Potential-Based Reward Shaping

1 INTRODUCTION

Reinforcement learning (RL) agents are able to solve many difficult problems, but are limited in their practical application by their poor sample complexity. The learning process is especially slow for long-sequence sparse-reward tasks. These kinds of tasks require complex sequences of actions before any non-zero reward is achieved. They often only reward a set of goal states, which are difficult to discover by chance. The RL agent therefore lacks meaningful reward feedback for most of its experiences with the environment and even if the agent finds a goal state it is challenging to assign the positive feedback correctly to the relevant parts of the required long action sequence.

One frequent strategy to help solve these kinds of problems is to use prior human knowledge in the form of symbolic models.

Proc. of the Adaptive and Learning Agents Workshop (ALA 2023), Cruz, Hayes, Wang, Yates (eds.), May 29-30, 2023, London, UK, https://alaworkshop2023.github.io/. 2023.

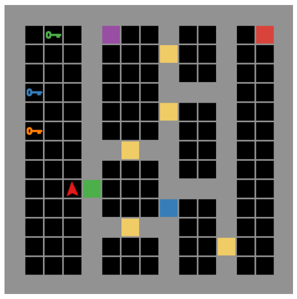


Figure 1: Visualization of the Household environment. The agent (red arrow) has to pick up the green key to unlock the green door, pick up the blue key to unlock the blue door, recharge itself on the charging station (purple square) and move into the goal (red square). The unlocked doors (yellow squares) can be traversed freely.

They can add structure to the task and guide the exploration of the environment. In hierarchical RL this knowledge is used to split the task into many easier to learn sub-tasks [14]. These approaches usually require some correct sequence of sub-tasks to be encoded in the symbolic model. Otherwise they would have to work with the complete combinatorial space of sub-task orderings, which in most tasks would be computationally infeasible. But even if the symbolic model fulfills the correctness requirement, the hierarchical RL approaches are limited by their task decomposition and lose their guarantee of convergence to the optimal policy if not all low-level primitive actions are also available at the higher levels, which is computationally expensive [4]. Another class of approaches is based on creating artificial reward signals from symbolic models. Directly redefining the reward function is difficult for humans as the expected scenarios do not have to exactly match the scenarios the agent will encounter and explore, which can lead to undesired side effects in the learned behavior [16]. Instead approaches with easier to define symbolic models like abstract high-level plans have been explored that automatically shape the reward based on the symbolic model [12, 20]. Nonetheless, these kinds of approaches are usually limited by (implicitly) assuming perfect symbolic plans.

In real world tasks imperfect knowledge will lead to incomplete and incorrect symbolic models. Take for example the household robot navigation task from [14] shown in Figure 1 . The robot has to pick up the green and blue keys to unlock the corresponding doors, charge itself at the charging station before moving into the red square. A human expert might not be able to differentiate the keys, but may know that any of the keys should be able to unlock the door. The expert could therefore specify to pick up any of the keys and leave it to the agent to try the keys. The agent would then have to learn from its experience, which key can be used to advance in the plan. The human expert might also lack knowledge of the initial charge of the agent and therefore does not know if the agent should charge itself after unlocking the first or second door, or if it has to charge itself at all to reach the goal. The plan could therefore contain the charging too many times, or only at the wrong positions in the plan, or not at all.

In this paper, we capture the high-level task structure through STRIPS plans automatically computed from abstract symbolic action representations designed by human experts. The STRIPS plans offer a natural way to decompose the task into a set of abstract symbolic subgoal states that are needed to reach the goal. We add an artificial dense reward signal to the reward function that rewards achieving these subgoals through potential-based reward shaping [27]. Intuitively, there is no incentive to incorporate incorrect knowledge as potential-based reward shaping can guarantee to not change the learned optimal policy and we can rely on the ability of RL to learn from experience to overcome imperfections of the abstract plan.

Although potential-based reward shaping is guaranteed to eventually converge to the optimal policy independent of the potential function and the quality of the knowledge encoded therein, we explore whether it can still accelerate the convergence of RL agents given incorrect or incomplete knowledge. In the hierarchical RL agent ASGRL [14] plan-based reward shaping (PBRS) [12] was used as a baseline and shown to be unreliable in accelerating the convergence of the RL agent when used with incomplete and incorrect knowledge. As a result of our work, we are the first to show that PBRS is in fact able to work with incomplete and incorrect knowledge on the same level as ASGRL if the used potential function is implemented correctly with respect to the specific requirements of PBRS. We then extend our experiments to more complex environments, where PBRS fails due to an inherent problem of scaling to plans with larger numbers of steps, as we prove theoretically. We propose a novel reformulation of the potential function that allows us to work with incomplete and incorrect plans of any length.

Our results show that our method improves over the performance of the state-of-the-art hierarchical RL agent ASGRL designed specifically to be able to use incomplete and incorrect knowledge.

We summarize our contributions as follows:

- We introduce the Exponential Plan-Based Reward Shaping (EPBRS) algorithm and demonstrate that its higher efficiency and better scalability compared to ASGRL [14], the current state-of-the-art technique to tackle the challenge of incomplete or incorrect plan knowledge.
- We empirically show that EPBRS is able to alleviate a wide range of types of incomplete or incorrect knowledge in its plan and is able to work with plans of any length.
- We prove the theoretical advantages of EPBRS for longer plans and verify them empirically on three different environments.

2 RELATED WORK

Prior human knowledge can be used for reinforcement learning to modify the reward function to specify tasks and to improve the sample efficiency. The knowledge can be encoded in many different representations like policy sketches [3], knowledge graphs [1, 2, 31], natural language instructions [10], or different specifications of (temporal) logic [6, 8, 19, 21, 23, 28].

Our work focuses on task knowledge specified as symbolic plans. Recent works in this direction include [9, 18, 20, 24, 25, 29, 30]. In contrast to these works we focus on incorrect and incomplete plans. In other previous works iterative refinement through continuous human feedback was used to overcome initial inaccuracies in the knowledge [5, 15]. We instead focus on the learning capabilities of RL agents to overcome the possible shortcomings of the human provided knowledge due to the costs of repeated human involvement.

Hierarchical RL agents are often used to solve long-sequence sparse-reward tasks due to their ability to simplify tasks by splitting them and solving the sub-tasks [17, 22, 25, 26]. Our method instead opts for non-hierarchical RL agents to avoid the large complexity of sub-task combinations when faced with incorrect knowledge.

3 BACKGROUND

3.1 Reinforcement Learning

A task in reinforcement learning is modeled as a Markov Decision Processes (MDP). An MDP M is defined as a tuple $M = (S, A, T, R, \gamma)$, where S and A are the state and action space respectively. $T(s, a, s') : S \times A \times S \rightarrow [0, 1]$ is the probability of reaching state s' from state s after executing action a. The reward function $R(s, a, s') : S \times A \times S \rightarrow \mathbb{R}$ assigns a numerical reward to a state transition from s to s' with respect to the executed action a. A policy $\pi(s, a) : S \times A \rightarrow [0, 1]$ defines how the agents should act in the environment through the probability distribution over all actions in every state. The decay factor $\gamma \leq 1$ is used to define the expected discounted return $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$ and the value function $V(s) = \mathbb{E}_{A_t \sim \pi(S_t)} [\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s]$. The reinforcement learning agent learns a policy that maximizes the expected discounted return, where the optimal policy has the maximum expected discounted return G^*.

3.2 Classical Planning

The additional domain knowledge input by a human user or domain expert is specified through STRIPS-like planning models. A planning model is a tuple $P = (F, A, I, G)$. The set of fluents F contains the symbolic high-level variables used to describe the environment states. Every state corresponds to a truth assignment of all the binary fluents such that the symbolic state space is defined as $S^P = 2^F$. We assume that the grounding of the model's fluents in the low-level environment states is given, as our method is agnostic to the implementation of the low-level fluent detection.

The set of actions A specifies the knowledge of the task structure splitting the task into high-level actions. An action $a \in A$ is a tuple $a = (prec, add, del)$, where the precondition of the action $prec \subseteq F$ defines the assignment of (a subset of) fluents needed to be able to execute the action. The add effects $add \subseteq F$ and delete effects $del \subseteq F$ specify the fluents that are respectively set true or false after executing the action.

The initial state $I \in S^P$ and the goal state $G \subseteq F$ are used to define the starting point and the target state of the plan. Without any loss of generality we assume that the goal state can be uniquely defined through a set of fluents.

We use the STRIPS-like planning model to automatically extract a high-level plan. The plan defines the possible sequences of actions that reach a valid goal state from the initial state. The sequential execution of the specified actions leads to a trajectory $\omega = (z_0, z_1, ..., z_g)$ of high-level states. Any high-level symbolic state, that is part of any goal-reaching trajectory ω_i in the plan, is a subgoal the agent should reach when solving a task and can be used in the low-level exploration as an intermediate landmark that can help reach the goal. In addition to the set of subgoals, it is possible to define a (partial) order over the subgoals. For any two subgoals s_i, s_j we write $s_i \prec s_j$ if s_i is achieved before s_j in all trajectories.

3.3 Potential-Based Reward Shaping

Long sequence, sparse reward RL tasks terminate with a reward of one once a sequence of specific high-level actions has been fulfilled. For all other transitions the agent gets a reward of zero. This lack of reward feedback creates very challenging tasks for RL agents due to the difficulties of assigning the reward correctly to the important parts within a long sequence of actions. The most notorious example is the Atari game *Montezuma's Revenge*, where the first non-zero reward is only obtained after collecting a key following a long sequence of actions. To get to the key, the agent has to navigate through the level climbing three ladders and a rope while avoiding to run into an enemy.

Reward shaping can help solve sparse reward tasks by utilizing external knowledge to add intermediate feedback into the reward function. Reward shaping creates a new shaped reward function $R'(s, a, s') = R(s, a, s') + F(s, a, s')$.

In potential-based reward shaping [27] the shaping function is defined by $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$, where $\Phi(s)$ is the so-called potential function. Potential-based reward shaping is the only way to define a reward shaping function that does not change the optimal policy, when no further assumptions on the transition and reward functions are made.

This does not hold in general for episodic MDPs with finite horizons such as the ones in this paper. As shown in [11] the expected return of the shaped agent is:

$$G_\Phi(s_0) = G(s_0) + \gamma^n \Phi(s_n) - \Phi(s_0)$$

$$G(s_0) = \sum_{i=0}^{n-1} \gamma^i R(s_i)$$

The potential of the first state is the same for all trajectories starting in the same state and therefore will not bias the optimal policy given the initial state. On the other hand, the potential of the terminal state reached after n steps depends on the chosen state trajectory. The number of steps until termination n can also change depending on the trajectory. With trajectories reaching a terminating (goal) state having a smaller n than n_{max} when the agent terminates after a fixed number of steps. The potential of the last step of any trajectory into a terminal state therefore has to equal zero, as not to change the optimal policy. Notably this also applies to the forced termination of an episode after the maximum number of steps has been reached.

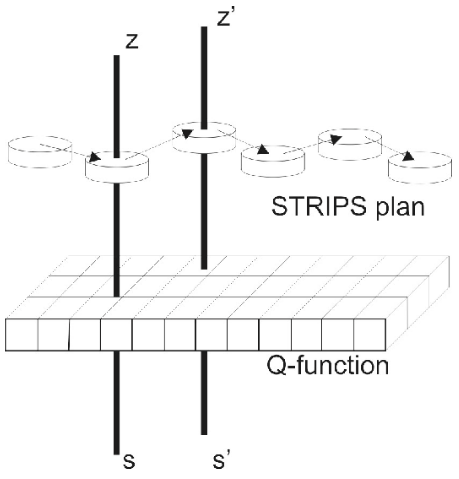


Figure 2: Schematic overview of Plan-Based Reward Shaping [7]

3.4 Plan-Based Reward Shaping

Plan-based reward shaping [12] is a method to use STRIPS plans for potential-based reward shaping. In plan-based reward shaping the potential function is defined as the highest step in the plan achieved so far. This creates a mapping from the low-level states to the symbolic steps in the plan as shown in Figure 2. The mapping is done by the *step* function, which is defined through the grounding function of the fluents, the subgoal set, and the partial order of subgoals. It extracts the assignment of the fluents for the given low-level state that are used in the planning model. The *step* function determines based on the fluent assignment if the current state matches any subgoal that has yet to be achieved. As the *step* function keeps track of the achieved steps, it can also be used for non-Markovian tasks. If the order for a subset of subgoals is not specified, their completion is allowed in any order. Otherwise the *step* count is strictly defined by the ordering information.

With this, the plan-based potential-function over the low-level states is defined as:

$$\Phi(s) = step(s) \tag{1}$$

Plan-Based Reward Shaping has to fulfill the following characteristics based on [13] for the shaped reward function to be useful for the agent:

$$R'(s, s') = R(s, s') + \gamma\Phi(s') - \Phi(s) > 0 \tag{2}$$

$$R'(s, s) = R(s, s) + \gamma\Phi(s) - \Phi(s) \leq 0 \tag{3}$$

$$R'(s, s_g) = R_G + \gamma\Phi(s_g) - \Phi(s) \geq 0 \tag{4}$$

where s is any state, s' is a non-terminal state reached from s with one action that fulfills a new landmark, s_g is a terminating goal state with a reward $R_G > 0$, and $R(s, s') = R(s, s) = 0$ in a sparse reward task. The shaping function should modify the reward function to simultaneously incentivize reaching a new subgoal and penalize remaining at the same step in the plan. Additionally the total reward for reaching the goal is required to stay positive. The step count and as such the potential is greater or equal to zero in any non-terminal state for plan-based reward shaping. As shown previously in Potential-Based Reward Shaping the potential of all terminal states including the goal has to be zero as not to change the optimal policy. The shaping function will therefore be negative when reaching the goal.

Simple arithmetic operations lead to the following inequalities for the plan-based potential function:

$$\gamma > \frac{\Phi(s)}{\Phi(s')} \tag{5}$$

$$(\gamma - 1)\Phi(s) \leq 0 \tag{6}$$

$$R_G \geq \Phi(s) \tag{7}$$

The first condition creates a limit on the number of steps that can be supported by plan-based reward shaping with regard to the reward decay factor γ. This condition starts to fail for large potential values in the latest steps of the plan and causes negative rewards when achieving a new landmark. Changing γ to allow for longer plans can also change the optimal policy and is therefore undesirable when utilizing potential-based reward shaping. Notice that the length limitation is only on the number of steps that can be specified in the plan, not on the number of actions in the environment. The second condition is trivially fulfilled for non-negative potential functions. The last condition limits the total value of the potential function to the reward value of reaching a terminal state. For the long-sequence sparse reward tasks that are the focus of this paper the only non-zero reward R_G is achieved for entering any goal state.

Adding the potential of zero for terminal states and the limit on the maximum value of the potential function leads to the complete definition of plan-based reward shaping:

$$\Phi(s) = \begin{cases} 0 & \text{, if s is a terminal state} \\ R_G \frac{step(s)}{|P|} & \text{, otherwise} \end{cases} \tag{8}$$

where $|P|$ is the total number of steps in the plan.

4 EXPONENTIAL PLAN-BASED REWARD SHAPING (EPBRS)

The main problem with the original definition of plan-based reward shaping is a result of the linear growth of its potential. The negative on-step rewards $F = \gamma\Phi(s) - \Phi(s)$ will increase in absolute value proportionally to the potential for any definition of a positive monotonic potential function. But the rewards for completing a new step in the plan $F = (\gamma - 1)\Phi(s) + \gamma c$ (where c is the constant difference between the potentials) do not increase but instead decrease in value with growing potential. Due to this, the agent will get a negative reward feedback sooner when exploring to reach another subgoal if the current best subgoal is later in the plan and if more subgoals precede it in the plan order.

We therefore propose to reformulate the plan-based potential function to grow exponentially with the number of completed steps. The exponential plan-based potential allows for both the negative on-step rewards and the positive on-subgoal rewards to increase proportionally with the number of fulfilled steps in the plan.

We propose to formulate the exponential plan-based potential as follows:

$$\Phi(s) = \begin{cases} 0 & \text{, if s is a terminal state} \\ R_G \frac{b^{step(s)}}{b^{|P|}} & \text{, otherwise} \end{cases} \tag{9}$$

where $b > 1$ is the constant base parameter, the step function is used as before to track the highest step in the plan achieved at the current state, and $|P|$ is the total number of steps in the plan.

Whenever a new landmark is reached, the agent gets an exponentially larger reward. The quotient normalizes the largest possible potential to prevent that too large negative shaped rewards from reaching terminal states drown out the external reward signal.

Applying our reformulation to the three conditions previously defined in Equation 2, Equation 3, and Equation 4 leads to:

$$b > \frac{1}{\gamma} \tag{10}$$

$$\gamma \leq 1 \tag{11}$$

$$R_G \geq \Phi(s) \tag{12}$$

The first condition no longer prohibits plans with too many steps. It instead sets a lower bound for the base parameter b to be used in the computations with regard to the decay factor γ. Equation 11 is trivially fulfilled as any positive potential function will lead to negative rewards when the potential does not change between steps. The last condition gives the reason for the normalization of the exponential potential with its maximum value and scaling it with the external reward awarded for reaching the goal.

On a more practical note, we now introduce a corollary to give an intuition for the impact of the chosen potential function on the exploration incentives. Due to our focus on monotonically increasing potential functions we expect positive rewards for achieving a higher potential in a new landmark and negative rewards for every step in which no new landmark is reached. Staying at one potential for an infinite time without achieving a new landmark will lead to a negative total shaping reward. In the following we focus on the exploration incentive created by the reward shaping. To be precise, the number of steps n an agent can take after achieving a new subgoal in the transition from state s_i to state s_{i+1} without reaching a new subgoal thereafter for which the decaying sum of shaping rewards is still positive:

$$\gamma\Phi(s_{i+1}) - \Phi(s_i) + \sum_{k=1}^{n} \gamma^k (\gamma\Phi(s_{i+1}) - \Phi(s_{i+1})) \geq 0 \tag{13}$$

Solving for n leads to the following general formula for any arbitrary potential-based reward shaping:

$$n = \log_\gamma \left(\frac{\Phi(s_i)}{\Phi(s_{i+1})} \right) - 1 \tag{14}$$

Using the *default* plan-based reward shaping we get the following formula:

$$n \leq \log_\gamma \left(\frac{\Phi(s_i)}{\Phi(s_i) + c} \right) - 1 \tag{15}$$

where c is the constant difference between two consecutive subgoals.

Using the *exponential* plan-based reward shaping we get the following formula:

$$n = \log_\gamma \left(\frac{1}{b} \right) - 1 \tag{16}$$

The default plan-based reward shaping leads to a decreasing number of steps n with an incentive to explore for later subgoals where the potentials are larger. The exponential plan-based reward shaping reformulation on the other hand offers the same fixed number of steps n independent of the current step in the plan. Instead the base factor for the calculation is proportional to the number of steps with a net positive cumulative shaped reward. This leads to the same exploration incentive for all steps in longer expert defined plans.

5 INCOMPLETE AND INCORRECT PLANS

We use abstract symbolic plans to encode the external human knowledge of how to solve the task. Requiring perfect plans is infeasible in practice. We rely on human experts to define the domain and task knowledge, who may not be able to capture the task description or environment completely, or may only posses knowledge of a closely related task. Additionally, more complex environments require more knowledge to be solved and can therefore require longer plans. To obtain a longer plan, one might have to include knowledge that is less certain to be correct introducing extra steps that are not needed to solve the task. Nonetheless, building on the knowledge should offer an advantage when learning how to act in an environment that does not offer much (reward) feedback by itself. Incorrect or incomplete definitions of the planning model only decrease the expected use of the encoded knowledge for solving the task.

There are many ways in which a plan can be incomplete or incorrect:

(1) The plan might lack important steps needed to solve the task, or might contain steps that are either not needed or are not achievable. While the first problem just decreases the amount of help the plan offers for solving the task, the second problem requires that planned steps can be skipped dynamically during training.
(2) The definition or detection of important steps can be incomplete. As such the fluents for a step may not only include correct useful states, but also some states that are useless for helping to solve the task. The agent then has to be able to either learn a diverse set of states for every step as in [14] or be able to learn from experience which states that fulfill the same subgoal are useful, which is the strategy used in our approach.
(3) Even if the plan contains all the correct and useful landmarks, the order defined by the plan might not be achievable in the task environment. As a result, the agent has to be able to dynamically skip some steps that are not (easily) achievable in the defined order and to learn them implicitly when they are missing in the plan.

When working with incomplete and incorrect plans *Exponential Plan-Based Reward Shaping* builds on the strengths of reinforcement learning. It can leverage its experience interacting with the environment to determine which steps help or hinder progressing the plan. Moreover it learns to dynamically skip steps that are useless for the completion of the task, and will eventually find the optimal policy under the convergence guarantee of reinforcement learning, even if the plan contains no useful landmarks at all. Notably, convergence to the optimal policy cannot be guaranteed in a hierarchical reinforcement learning system like ASGRL [14] unless all low-level primitive actions are also available at the higher levels, which is computationally expensive [4].

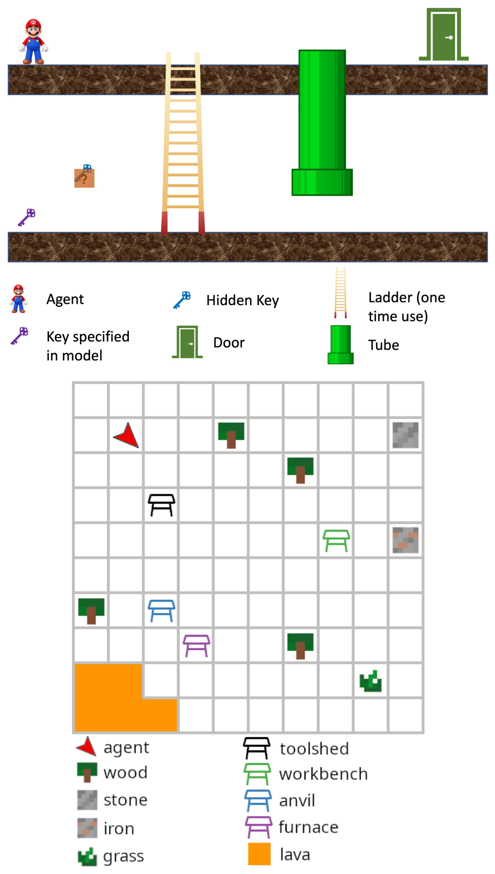


Figure 3: Visualization of the Mario Environment [14] and the Minecraft Environment

Additionally, we do not require that the elements of the planning model accurately match their corresponding parts in the low-level task. The detector for the fluents does not have to be perfect. The high-level plan actions do not have to be executable in the low-level environment. The initial and goal states of the planning model do not have to match the initial and goal states of the MDP.

6 EXPERIMENTAL SETUP

6.1 Environments

We empirically evaluate our method on three different environments. The shorter, less complex task will be referred to as `mario` environment and mirrors the respective environment used in [14]. In addition, we also use two more complex tasks referred to as `household` environment and `minecraft` environment. The more complex tasks allow for longer and more complex plans and are used to verify the scalability of our method with regard to the length of the plan. All tested methods have access to the same plans and therefore to the same external task knowledge.

6.1.1 The Mario environment. This environment is a representative example of a sparse-reward task. In this modified version the agent has to open the door on the upper level with two keys from the lower level, one of which is hidden in a box. The agent can move between the levels either on a ladder that breaks after the first use, or through a tube that can only be traversed downwards. The optimal sequence of actions would therefore be to move down through the tube, pick up both keys, and move back up on the ladder to unlock and enter the door.

The expert lacks knowledge of the tube and the hidden key. The plan therefore contains instructions to only use the ladder to move between the levels, which would lead to being stuck on the lower level, and to pick up only the visible key. The agent then has to overcome the actively harmful high-level instruction and learn the missing symbolic states from experience.

6.1.2 The Household environment. The `household` environment shown in Figure 1 originates from the implementation of [14]. A robot has to navigate through multiple rooms to reach the goal state. First, it has to pick up the green key to unlock the green door, then pick up the blue key to unlock the blue door, and recharge itself on the charging station before moving into the goal.

We consider two plans for the `household` environment. The first plan correctly specifies the steps and their order, but the human expert does not know that only one of the keys can unlock each door and specifies a plan where any of the keys can be used. The second plan is an extension of the first plan with ten additional useless steps in the first room, which guide the agent to initially move downwards, then left and back upwards, stopping in front of the orange key that cannot be used to unlock any door. All of these additional steps are not needed to complete the task and will be skipped entirely in optimal trajectories. This plan is used to highlight scaling problems with regards to the length of an incorrect plan. We will refer to the environment in combination with the second plan as `over-planned household`.

6.1.3 The Minecraft environment. This environment is based on the Minecraft-inspired 2D crafting-game used in [3]. We use a more challenging definition closer to the resource progression originally used in the game Minecraft.

The agent can collect wood, stone, and iron, but can only carry one of each item at once in its inventory. The goal of the agent is to collect grass for which it gets a reward of one. This requires the agent to progress through a sequence of tools: First crafting a wooden pickaxe to collect stone, then a stone pickaxe to collect iron ore, and lastly a pair of shears in order to safely harvest grass without destroying it. The environment contains four different crafting stations: At the workbench the agent can turn wood into planks. The tool shed can be used to turn planks into sticks, which are needed to craft the pickaxes. The anvil can be used to craft all of the tools, and the furnace can be fueled with wood to smelt iron ore into an iron ingot, which is needed to craft the shears. If the agent at any point enters the lava lake it will die and the episode will terminate immediately with a reward of zero.

The human expert knows all the required abstract steps, but does not know that the inventory of the agent is limited to one of each item and only specifies the following order of steps. The agent should collect the four needed pieces of wood first, then craft the three needed planks, and then craft the two needed sticks. The agent will have to either learn extra steps to get rid of the current version of the item in its inventory by completing later crafting steps before obtaining another version, or skip the repetitions and learn them implicitly when they are needed later in the plan. After these incorrectly ordered steps the remaining steps are ordered correctly and can be achieved in their order in the environment.

6.2 Algorithms

6.2.1 Q-Learning. This baseline implements the standard, unmodified tabular Q-Learning algorithm and learns directly from the sparse unmodified reward signal.

6.2.2 ASGRL. This baseline uses the implementation of [14]. It is a state-of-the-art hierarchical reinforcement learning algorithm. ASGRL is designed to alleviate the sparse reward signals by utilizing so-called landmark sub-goals to split the task into easier sub-tasks. It then learns to solve the task by utilizing multiple policies that are able to move the agent from one landmark to the next. A diversity objective is used to incentivize a diverse set of states fulfilling a landmark to overcome incomplete knowledge. As in the original paper we use the history-based meta-state representation for ASGRL. We use their non-curriculum setting as it has shown superior performance in all of our experiments compared with curriculum ASGRL. We use $k = 3$ for ASGRL in all experiments for all (sub-)policies to be able to cover all possible landmark states.

6.2.3 Plan-based Reward Shaping. This baseline is an implementation of plan-based reward shaping as originally described in [12] in combination with the Q-Learning baseline algorithm. We use the modified potential function as defined in equation 8 with the same $step$ function as our approach.

6.2.4 Exponential PBRS. Our approach.

6.3 Hyperparameters

The hyperparameters for all algorithms are based on the ones used in [14].

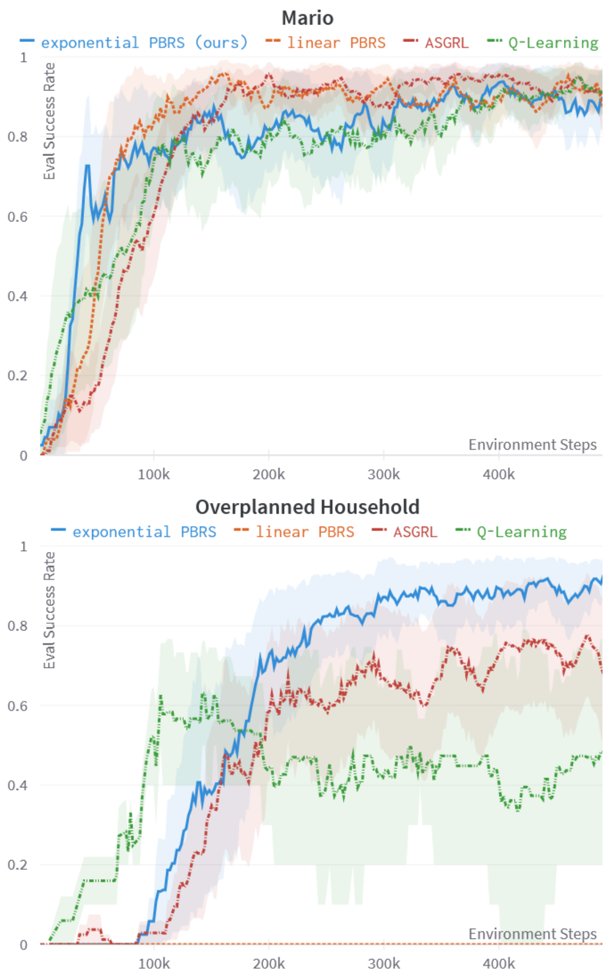
We use $\gamma = 0.99$ for all environments. The algorithms have a limit of 2,000 steps per episode before being truncated, the individual sub-policies in ASGRL have a limit of 700 steps per episode each, and the training is limited to 500,000 steps total for all experiments. We use a replay buffer of successful trajectories to sample uniformly from during training to focus important experiences.

For the Q-Learning agents that are used in all baselines we use ϵ-greedy to balance exploration and exploitation. We use the annealing on success of both ϵ and the learning rate used in ASGRL for all our algorithms.

For the Q-Learning agents the value of ϵ is annealed from 1.0 to 0.05 by a factor of 0.95 whenever the final goal state is reached and the learning rate is annealed from 1.0 to 0.1 by a factor of 0.95 whenever the final goal state is reached. The same parameters are used for each skill-policy in ASGRL with individual updates on skill success. The ϵ_2 for the ASGRL meta-controller is annealed from 1.0 to 0.05 by a factor of 0.9 whenever the final goal state is reached.

7 RESULTS

We evaluate all algorithms on all four tasks with regard to their ability to solve the long-sequence sparse-reward tasks and their sample efficiency. We measure the success rate and sample efficiency during the training. Each algorithm was evaluated every



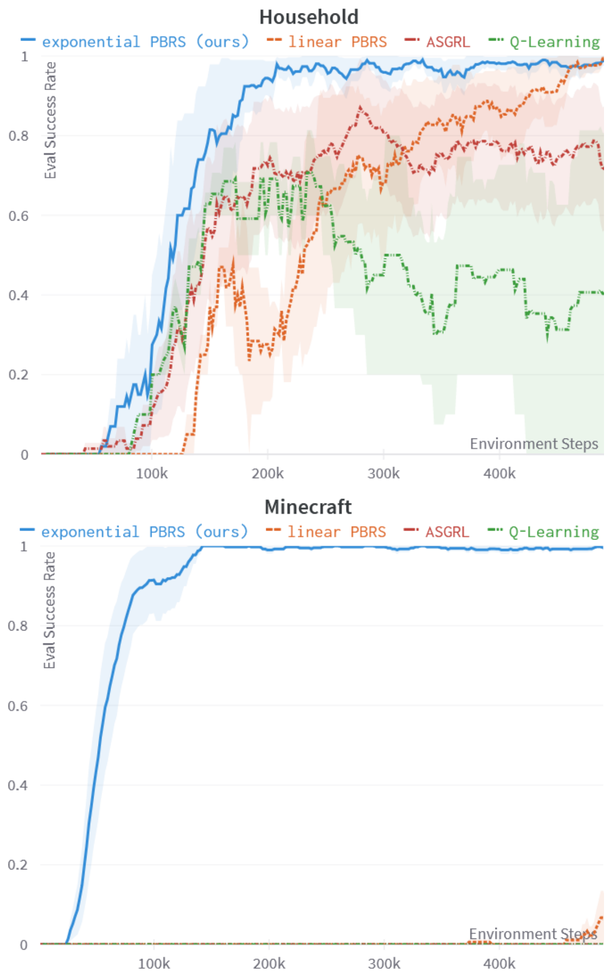


Figure 4: Comparison of tested algorithms for the respective environments. Solid lines represent the mean of ten runs per algorithm. The shaded areas show the standard error of the mean.

five training episodes on whether they are able to reach the goal with a low exploration probability $\epsilon = 0.05$. We report the mean and standard error of the mean of ten repeated runs with different seeds for every algorithm in every experiment.

7.1 The Mario Environment

The `mario` environment is the simplest environment we experiment with. Contrary to the results of [14], Q-Learning already performs well in the Mario environment if the same annealing of the learning rate and ϵ as for ASGRL is used instead of a fixed decay after every episode. The environment is small enough that the Q-Learning agents are able to find the goal state reliably by chance within the first 250,000 steps. The success rate and convergence speed of the plan-based reward shaping methods and ASGRL are usually slightly better than for the pure Q-Learning baseline, as they are able to leverage their high-level plan to guide the exploration. Both PBRS and ASGRL are able to learn the missing steps in their plan through their Q-Learning agents.

7.2 The Household Environment

The `household` environment is larger and therefore more difficult to solve. The Q-Learning baseline without any external symbolic knowledge is only able to solve the task reliably in 40% of the runs at the end of the computation budget of 500,000 steps. Initially at around 200,000 steps, more of the Q-Learning agents are able to find the goal in some evaluation runs, but they struggle with assigning the reward correctly to the relevant parts of their action sequence. Due to this the Q-Learning agents end up with a large standard error of the mean as the individual runs either learn to reliably solve the task or are unable to solve the task at all. The ASGRL agents are able to solve the task more reliably than the Q-Learning baseline. The decomposition into sub-tasks helps to stabilize the learning by augmenting the reward function with the subgoal information, which helps to find the relevant parts of the action sequence. The plan-based reward shaping methods are both able to reliably solve the task at the end of the training. The exponential Plan-Based Reward Shaping converges faster, as it is

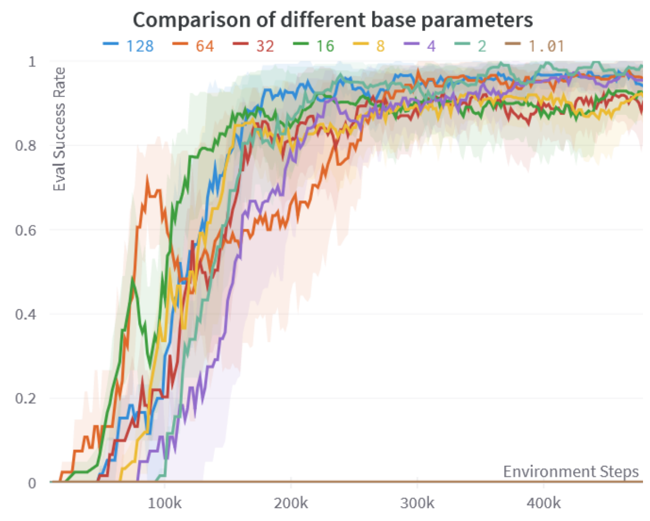


Figure 5: Comparison of the different values for the base parameter b used in exponential PBRS in the household environment. The solid lines show the means of ten repeated runs with different seeds and the shaded areas display the standard error of the mean.

better suited for the longer plan. The linear Plan-Based Reward Shaping initially avoids the later steps in the plan, but is able to still solve the task due to the theoretical guarantees of potential-based reward shaping. All of the plan-based agents are able use the plan to improve their evaluation success rate even with the ambiguous definition of the needed keys to unlock the doors.

7.3 The Over-Planned Household Environment

When we use the extended plan in the over-planned household environment, ASGRL converges slightly slower due to the additional learning to achieve the extra steps. In contrast to the plan-based reward shaping agents ASGRL is biased to fulfill as many subgoals as possible before reaching the goal and will not learn to ignore the useless extra steps at the beginning of the plan. The linear plan-based reward shaping has an evaluation success rate of zero over the entire training duration as it is does not scale well to plans with many steps. The later steps in the longer plan have a shorter horizon where they offer a positive exploration reward. Therefore, the linear plan-based reward shaping instead starts to punish reaching the later landmarks and tries to avoid them. The exponential plan-based reward shaping is able to scale to plans with many steps and reaches the best evaluation success rate during training of all tested algorithms. Moreover it is guaranteed to converge towards the optimal policy ignoring the useless added steps in the plan.

7.4 The Minecraft Environment

The Minecraft environment is the most difficult task we experiment with. The lava lake creates an easy to reach way to terminate the episode without any reward feedback. Random exploration is therefore much more challenging as the agent does not only have to achieve a long sequence of specific high-level actions, but also has to avoid entering the lava lake by chance. The Q-Learning agent is therefore unable to reach the final goal in any evaluation runs during the entire training. The ASGRL agent requires the plan to be valid for at least one trajectory that reaches the goal to avoid having to explore all combinations of subgoals. ASGRL is therefore unable to succeed at all in the Minecraft environment as the plan encodes an order of steps that is not achievable in the environment. The exponential Plan-Based Reward Shaping agent is the only tested agent that is able to use the provided incorrect plan to be able to learn to solve the task. It is able to skip the early steps that cannot be achieved in the environment and is able to instead learn them implicitly when they are needed to progress in the plan. Notably, all ten trial runs are able to reliably solve the task after 150,000 steps with an evaluation success rate close to 1.0. The linear Plan-Based Reward Shaping agent once again struggles with the length of the plan and is not able to solve the plan. Only at the end of the training one of the runs is able to solve the task unreliably. The agent still appears to be able to make some use of the incorrect and long plan to simplify solving the task.

7.5 Impact of Base Parameter b

First we evaluate the choice of the base parameter b for the calculation of the exponential in our method on the household environment. For this we run ten repeated experiments per value of b and track the success rate of the policies during training when evaluated with a low exploration probability $\epsilon = 0.05$. The results are shown in figure 5. Most importantly our formal limit in Equation 10 also holds empirically. Values for $b \leq 1/\gamma$, in this case $1.\overline{01}$, are not able to solve the task at all within our computation budget of 500,000 steps. Choosing $b > 1/\gamma$ for the other runs leads to fast convergence to evaluation success rates between 0.8 and 1.0. The different values for b show little difference in convergence speed, although larger values tend to perform better. Overall the differences for the runs with $b > 1/\gamma$ are small enough to to be related to stochastic noise introduced by different random seeds and different random explorations of the ϵ-greedy RL algorithm.

We therefore choose $b = 32$ for all runs of Exponential Plan-Based Reward Shaping in all experiments.

8 CONCLUSION

Reinforcement learning in long-sequence sparse reward tasks can benefit from human knowledge even if the knowledge is incomplete or partially incorrect. In this paper, we are the first to show that plan-based reward shaping (PBRS) is able to work with incomplete and incorrect plans if the used potential function fulfills the set of requirements we formally derived. We prove that PBRS has an inherent theoretical flaw that prevents it from scaling to more complex environments with more challenging tasks that require longer plans. We propose a new formulation of plan-based reward shaping that can improve reward feedback using longer, incomplete and incorrect plans in more complex environments. We show its improved effectiveness for different kinds of incomplete and incorrect plans in multiple environments and when scaling up to more complex tasks with longer plans outperforming the state-of-the-art. Our method provides robust performance gains in combination with all different kinds of incomplete or incorrect knowledge.

REFERENCES

[1] Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and William L. Hamilton. 2020. Learning Dynamic Belief Graphs to Generalize on Text-Based Games. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 256, 13 pages.

[2] Prithviraj Ammanabrolu and Mark Riedl. 2019. Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 3557–3565. https://doi.org/10.18653/v1/N19-1358

[3] Jacob Andreas, Dan Klein, and Sergey Levine. 2017. Modular Multitask Reinforcement Learning with Policy Sketches. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17)*. JMLR.org, 166–175.

[4] Andrew G Barto and Sridhar Mahadevan. 2003. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems* 13, 1 (2003), 41–77.

[5] Chandrayee Basu, Mukesh Singhal, and Anca D. Dragan. 2018. Learning from Richer Human Guidance: Augmenting Comparison-Based Learning with Feature Queries. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction* (Chicago, IL, USA) *(HRI '18)*. Association for Computing Machinery, New York, NY, USA, 132–140. https://doi.org/10.1145/3171221.3171284

[6] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. 2021. Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications. *Proceedings of the International Conference on Automated Planning and Scheduling* 29, 1 (May 2021), 128–136. https://doi.org/10.1609/icaps.v29i1.3549

[7] Kyriakos Efthymiadis and Daniel Kudenko. 2015. Knowledge Revision for Reinforcement Learning with Abstract MDPs. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems* (Istanbul, Turkey) *(AAMAS '15)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 763–770.

[8] Mahmoud Elbarbari, Florent Delgrange, Ivo Vervlimmeren, Kyriakos Efthymiadis, Bram Vanderborght, and Ann Nowe. 2022. A framework for flexibly guiding learning agents. *Neural Computing and Applications* (06 2022). https://doi.org/10.1007/s00521-022-07396-x

[9] Clement Gehring, Masataro Asai, Rohan Chitnis, Tom Silver, Leslie Pack Kaelbling, Shirin Sohrabi, and Michael Katz. 2022. Reinforcement Learning for Classical Planning: Viewing Heuristics as Dense Reward Generators. In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, Akshat Kumar, Sylvie Thiébaux, Pradeep Varakantham, and William Yeoh (Eds.). AAAI Press, 588–596. https://ojs.aaai.org/index.php/ICAPS/article/view/19846

[10] Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. 2019. Using Natural Language for Reward Shaping in Reinforcement Learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 2385–2391. https://doi.org/10.24963/ijcai.2019/331

[11] Marek Grzes. 2017. Reward Shaping in Episodic Reinforcement Learning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems* (São Paulo, Brazil) *(AAMAS '17)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 565–573.

[12] Marek Grzes and Daniel Kudenko. 2008. Plan-based reward shaping for reinforcement learning. In *2008 4th International IEEE Conference Intelligent Systems*, Vol. 2. 10–22–10–29. https://doi.org/10.1109/IS.2008.4670492

[13] Marek Grzes and Daniel Kudenko. 2009. Theoretical and Empirical Analysis of Reward Shaping in Reinforcement Learning. In *2009 International Conference on Machine Learning and Applications*. 337–344. https://doi.org/10.1109/ICMLA.2009.33

[14] Lin Guan, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Leveraging Approximate Symbolic Models for Reinforcement Learning via Skill Diversity. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.). PMLR, 7949–7967. https://proceedings.mlr.press/v162/guan22c.html

[15] Lin Guan, Mudit Verma, and Subbarao Kambhampati. 2020. Explanation Augmented Feedback in Human-in-the-Loop Reinforcement Learning. *CoRR* abs/2006.14804 (2020). arXiv:2006.14804 https://arxiv.org/abs/2006.14804

[16] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart Russell, and Anca D. Dragan. 2017. Inverse Reward Design. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6768–6777.

[17] Danijar Hafner, Kuang-Huei Lee, Ian Fischer, and Pieter Abbeel. 2022. Deep Hierarchical Planning from Pixels. https://doi.org/10.48550/ARXIV.2206.04114

[18] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. 2021. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 9 (May 2021), 7647–7656. https://doi.org/10.1609/aaai.v35i9.16935

[19] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. 2018. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 2107–2116. https://proceedings.mlr.press/v80/icarte18a.html

[20] León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A. McIlraith. 2020. Symbolic Plans as High-Level Instructions for Reinforcement Learning. *Proceedings of the International Conference on Automated Planning and Scheduling* 30, 1 (Jun. 2020), 540–550. https://doi.org/10.1609/icaps.v30i1.6750

[21] Yuqian Jiang, Sudarshanan Bharadwaj, Bo Wu, Rishi Shah, Ufuk Topcu, and Peter Stone. 2020. Temporal-Logic-Based Reward Shaping for Continuing Learning Tasks. *CoRR* abs/2007.01498 (2020). arXiv:2007.01498 https://arxiv.org/abs/2007.01498

[22] Mu Jin, Zhihao Ma, Kebing Jin, Hankz Hankui Zhuo, Chen Chen, and Chao Yu. 2022. Creativity of AI: Automatic Symbolic Option Discovery for Facilitating Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 36, 6 (Jun. 2022), 7042–7050. https://doi.org/10.1609/aaai.v36i6.20663

[23] Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. 2021. Compositional Reinforcement Learning from Logical Specifications. *CoRR* abs/2106.13906 (2021). arXiv:2106.13906 https://arxiv.org/abs/2106.13906

[24] Harsha Kokel, Arjun Manoharan, Sriraam Natarajan, Balaraman Ravindran, and Prasad Tadepalli. 2021. RePReL: Integrating Relational Planning and Reinforcement Learning for Effective Abstraction. *Proceedings of the International Conference on Automated Planning and Scheduling* 31, 1 (May 2021), 533–541. https://doi.org/10.1609/icaps.v31i1.16001

[25] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. 2019. SDRL: Interpretable and Data-Efficient Deep Reinforcement Learning Leveraging Symbolic Planning. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence* (Honolulu, Hawaii, USA) *(AAAI'19/IAAI'19/EAAI'19)*. AAAI Press, Article 365, 8 pages. https://doi.org/10.1609/aaai.v33i01.33012970

[26] Ludovico Mitchener, David Tuckey, Matthew Crosby, and Alessandra Russo. 2022. Detect, Understand, Act: A Neuro-Symbolic Hierarchical Reinforcement Learning Framework (Extended Abstract). In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, Lud De Raedt (Ed.). International Joint Conferences on Artificial Intelligence Organization, 5314–5318. https://doi.org/10.24963/ijcai.2022/742 Sister Conferences Best Papers.

[27] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML '99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 278–287.

[28] Mathieu Tuli, Andrew C Li, Pashootan Vaezipoor, Toryn Q. Klassen, Scott Sanner, and Sheila A. McIlraith. 2022. Learning to Follow Instructions in Text-Based Games. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=StlwkcFsjaZ

[29] Fangkai Yang, Daoming Lyu, Bo Liu, and Steven Gustafson. 2018. PEORL: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (Stockholm, Sweden) *(IJCAI'18)*. AAAI Press, 4860–4866.

[30] Ruohan Zhang, Faraz Torabi, Lin Guan, Dana H. Ballard, and Peter Stone. 2019. Leveraging Human Guidance for Deep Reinforcement Learning Tasks. *CoRR* abs/1909.09906 (2019). arXiv:1909.09906 http://arxiv.org/abs/1909.09906

[31] Sijin Zhou, Xinyi Dai, Haokun Chen, Weinan Zhang, Kan Ren, Ruiming Tang, Xiuqiang He, and Yong Yu. 2020. Interactive Recommender System via Knowledge Graph-Enhanced Reinforcement Learning. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 179–188. https://doi.org/10.1145/3397271.3401174