Learning Parameterized Policies for Planning Annotated RL

Harsha Kokel, Junkyu Lee, Michael Katz, Shirin Sohrabi

IBM Research
{harsha.kokel,junkyu.lee,michael.katz1}@ibm.com,ssohrab@us.ibm.com

Abstract

Recently, several approaches have utilized AI planning in the context of hierarchical reinforcement learning. These methods employ planning operator descriptions to establish options for acquiring primitive or low-level skills. By employing hierarchical decomposition through operators, these approaches offer notable benefits during training, such as enhanced sample efficiency, as well as during evaluation, with improved generalization across different yet related tasks. In this study, we introduce a novel approach for defining parameterized options using operator descriptions. Our empirical evaluations conducted on the mini-grid domain demonstrate that the proposed approach not only enhances sample efficiency but also overcomes certain limitations associated with generalization capabilities.

1 Introduction

Many real-world domains involving sequential decision-making exhibit a compositional nature. This implies that these domains possess an inherent hierarchy that allows decision-makers to abstract certain details and focus on making high-level decisions. For instance, let's consider the task of driving from San Jose Airport to San Francisco Airport. By abstracting away low-level details such as traffic signals, road conditions, and potential roadblocks, we can create a high-level plan offline. This plan might involve getting on US-101 N from Airport Blvd and Airport Pkwy, following US-101 N to Exit 422 in San Mateo County, and ultimately following signs for the domestic terminal. However, executing this plan at a low-level necessitates specific skills and online planning.

It has been observed that the abstracted high-level transition system can often be described symbolically, commonly using the Planning Domain Description Language (PDDL) (McDermott 2000). This ease of describing high-level symbolic transitions, coupled with the ability of reinforcement learning (RL) agents to acquire low-level skills through exploration, has inspired various recent works to propose two-level frameworks (Lee et al. 2022; Kokel et al. 2022b,a, 2021; Silver et al. 2022; Illanes et al. 2020; Sarathy et al. 2021; Jin et al. 2022; Lyu et al. 2019; Eppe, Nguyen, and Wermter 2019). These frameworks utilize RL agents to learn low-level skills while employing planners to generate high-level sub-goals. The primary advantage these frameworks aim to offer is the transferability of learned skills. Skills acquired for one task can be applied to another task, enabling efficient skill reuse and adaptation.

While such frameworks have shown some success in transferring to another task, their ability to generalize to different objects is limited. In this work, we focus on extending the generalization ability to problems in same domain with different objects. To do this, we introduce a novel approach for defining parameterized options using planning operator descriptions. Our initial experiments on minigrid domain demonstrates that the proposed approach can not only improve generalization ability to different objects but also enhance sample efficiency.

2 Background

2.1 Reinforcement Learning

A goal-oriented sequential decision-making problem can be represented as a goal-oriented Markov decision process (GMDP) $\langle S, A, T, R, \gamma, G\rangle$. Here, S denotes the set of states, A represents the set of actions, $T : S \times A \times S \mapsto [0, 1]$ represents the transition function, $R : S \times A \times S \mapsto \mathbb{R}$ represents the reward function, $\gamma \in (0, 1)$ represents the discount factor, and G represents the set of goal states. The objective for the agent is to interact with the environment and find an optimal policy $\pi : S \times G \times A \mapsto [0, 1]$ that maximizes the expected cumulative discounted reward.

2.2 Options framework

When the action space is excessively large and/or the state space is impractical to enumerate, RL agents experience the curse of dimensionality. By introducing temporally extended actions in hierarchical RL (Dietterich 1998; Sutton, Precup, and Singh 1998; Parr and Russell 1998), researchers aimed to leverage the ability of humans to simplify problems by breaking them down into manageable sub-problems or abstractions. Temporally extended actions allows agents to operate at multiple levels of abstraction. This promotes more efficient exploration, faster learning, and improved generalization to new situations. By decomposing complex actions into a hierarchy, agents can tackle large problems in a more

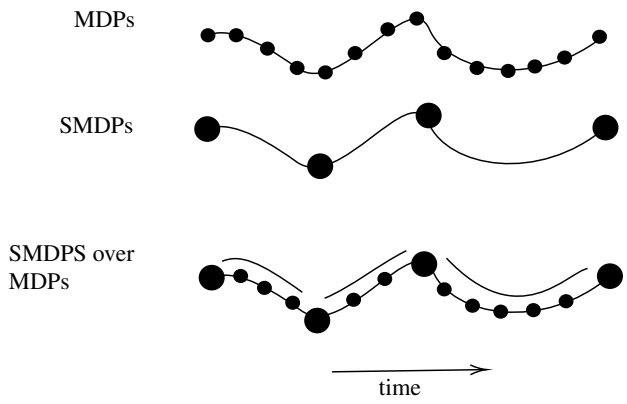


Figure 1: Comparison of actions in MDPs, SMDPs, and HRL

organized and structured manner, leading to better performance and scalability in RL tasks.

MDPs model time as discrete steps, where actions persist for only a single step. In contrast, Semi-MDPs (SMDPs) introduce temporally extended actions of varying lengths over continuous time, as depicted in the first two trajectories of Figure 1. By incorporating temporally extended actions, HRL approaches augment the primitive action space of MDPs, combining elements of both MDPs and SMDPs, as illustrated in the last trajectory of Figure 1.

The Options framework introduced by Sutton, Precup, and Singh (1998) provides a modeling approach for temporally extended actions known as *options*. An option $p = \langle I_p, \beta_p, \pi_p \rangle$ consists of three components: a set of states where the option can be initiated (I_p), a termination condition ($\beta_p : S \mapsto [0, 1]$) that determines the probability of option termination, and an option policy (π_p). A policy is learned for each option. At each step, the agent decides between taking a primitive (single-step) action or initiating a temporally extended option. Once an option is initiated, the agent employs the corresponding option policy to determine actions until the termination condition is satisfied. The Options framework offers a notable advantage in transfer learning. A skill learned to accomplish one high-level task can be applied to another high-level task, promoting knowledge reuse. However, a limitation of the options framework is that it learns a single policy for a specific termination condition. In order to effectively generalize across different tasks, the option policy should be adaptable to changing termination conditions.

2.3 Planning

A typed *first-order language* $\mathcal{L} = \langle \mathcal{P}, \mathcal{B}, \mathcal{T}, \mathcal{V} \rangle$ comprises a finite set of predicates $\mathcal{P}$, objects $\mathcal{B}$, types $\mathcal{T}$, and free variables. A *substitution* θ maps variables to objects. An *atom* is a predicate symbol followed by a parenthesized list of terms, e.g., $\texttt{predicate(term}_1\texttt{, term}_2\texttt{, } \cdots\texttt{)}$. A term can be a variable or an object. A *literal* is either an atom or the negation of an atom. An atom is considered a *ground atom* if all its terms are objects; otherwise, it is referred to as a *lifted atom*. A lifted atom can be grounded by applying a substitution. For instance, with the substitution $\alpha = \{X/\texttt{p1}\}$, the atom $a = p(X)$ is grounded to $a(\alpha) = p(\texttt{p1})$. We use the symbol $\models$ to denote entailment. For example, $A \models B$ indicates that in every possible world where A holds `True`, B also holds `True`.

A planning domain $\mathbf{D} = \langle \mathcal{L}, \mathcal{O} \rangle$ consists of a first-order language $\mathcal{L}$ and a finite set of schematic operators $\mathcal{O}$. A *schematic operator* is defined as $o = \langle h(o), pre(o), add(o), del(o) \rangle$, which includes a lifted atom, $h(o)$, referred to as the *head*; a first-order formula representing *preconditions* $pre(o)$; and two disjoint sets of atoms, $add(o)$ and $del(o)$, representing the positive (add) and negative (delete) *effects* of executing the operator. All terms appearing in the literals of $pre(o)$, $add(o)$, and $del(o)$ are also arguments of $h(o)$. Thus, a schematic operator can be grounded by substituting the head atom, resulting in $h(o)\alpha$. A possibly empty subset of literals in the preconditions that do not appear in the effects (positive or negative) is called the *prevail condition*, denoted as $prv(o)$. A *ground operator* $o(\alpha)$ can be applied in a state s if a substitution α satisfies the precondition $pre(o)$ in s, i.e., $s \models pre(o(\alpha))$. When the ground operator $o(\alpha)$ is applied, the state s transitions to another state $s' = (s \backslash del(o(\alpha))) \cup add(o(\alpha))$.

A classical planning task is defined as a tuple $\Pi = \langle \mathbf{D}, s_0, \mathcal{G} \rangle = \langle \mathcal{P}, \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{O}, s_0, \mathcal{G} \rangle$, where s_0 represents the initial state, and goal $\mathcal{G}$ is specified as a conjunction of literals. A plan for the planning task is a sequence of ground operators that, when executed in the initial state s_0, leads to a state satisfying $\mathcal{G}$.

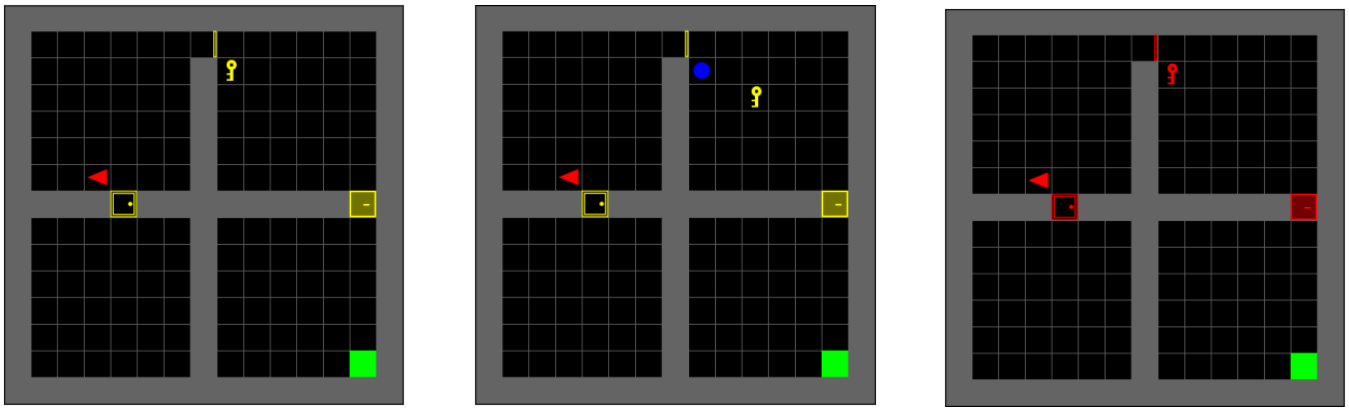
2.4 Planning annotated RL task

In the Planning annotated RL (PaRL) framework proposed by Lee et al. (2022), a PaRL task $E = \langle M, \Pi, L \rangle$ is defined over an GMDP M, a planning task Π, and a mapping $L : S \mapsto \bar{S}$ from the states of the MDP to planning states. In the PaRL framework, an option $p = \langle I_p, \beta_p, \pi_p \rangle$ is defined for each *grounded* operator $o(\alpha)$ in Π as follows:

1. Initiation set: $I_p = \{s \in S | L(s) \models pre(o(\alpha))\}$
2. Termination set: $\beta_p = \{s \in S | L(s) \models prv(o(\alpha)) \cup \mathit{eff}(o(\alpha))\}$.

Additionally, a single *goal option* is defined to achieve the goal (G) of the GMDP after the goal condition ($\mathcal{G}$) in the planning task is satisfied. The goal option is represented as $\langle s \in S | L(s) \models \mathcal{G}, G, \pi_g \rangle$, where the option's initiation set consists of states that satisfy the goal condition, and the policy π_g is trained to accomplish the GMDP goal. The policies for these options are trained using an online learning approach. Specifically, the GMDP of the PaRL task is decomposed into separate MDPs for each option, and the policies for these options are trained. During the training process, an intrinsic reward is incorporated, which encourages adherence to frame constraints.

A PaRL agent that has been trained on a locked door-key environment within the mini-grid domain, as shown in Figure 2a exhibits efficient generalization to a locked door-key environment with an obstacle, depicted in Figure 2b. However, the agent encounters difficulties



(a) (b) (c)

Figure 2: Three variants of minigrid domain with 4 rooms. The agent is represented with a red triangle. Goal is to navigate the agent to the goal location, represented as a green cell. (a) A locked door-key environment. (b) A locked door-key environment with the blue ball as an obstacle. (c) A locked door-key environment with red keys.

when trying to generalize to a locked door-key environment with a red key. To comprehend this limitation, let's consider the list of grounded operators defined in the initial environment. This list would include operators such as `(pick yellow_key room_0_0)`, `(pick yellow_key room_0_1)`, `(unlock yellow_key door_1)`, and so on. However, the operator `(pick red_key room_0_0)` does not exist in the initial environment. Consequently, no option has been trained specifically for this operation. As a result, the generalization capability of the PaRL agents is constrained to the objects present in the training environment. In this work, our objective is to address this limitation and overcome it by enhancing the generalization capability of PaRL agents.

3 Parameterized PaRL

The primary objective of this work is to train an agent capable of generalizing to various tasks that may involve objects that have not been seen during training. To provide clarity to the concept of "generalization to different tasks," we define a domain-constrained distribution of PaRL tasks and specifically emphasize the agent's ability to generalize within that distribution. By focusing on tasks within this defined distribution, we aim to enhance the agent's capacity to adapt and perform well in novel scenarios that share the same domain constraints.

Definition 1 *A **domain-constrained distribution** of PaRL task $\mathcal{E}$ is a distribution of PaRL tasks in which all the tasks:*

1. *Share a common set of operators $\mathcal{O}$ and actions A.*
2. *Share a common superset of predicates $\mathcal{P}$, objects $\mathcal{B}$, and types $\mathcal{T}$.*

It is important to note that the predicates, objects, and types within each task of the domain-constrained distribution may vary. For instance, in the mini-grid task depicted in Figure 2, the task shown in Figure 2a includes a `yellow-key` object, while the task in Figure 2c includes a `red-key` object. Furthermore, while the task in Figure 2a does not feature any object of type `ball`, the task in Figure 2b does. In order to generalize across such a distribution, it is necessary to learn a policy that can accommodate this variability. To achieve this, we employ the approach of learning parameterized options.

Definition 2 *A **parameterized option** is defined as a 4-tuple $p = \langle I_p, \theta_p, \beta_p, \pi_p \rangle$, where:*

- *I_p is the set of states where the option can be initiated, with $I_p \subset 2^S$.*
- *θ_p represents the parameter space associated with the option.*
- *$\beta_p : S \times S \times \theta_p \mapsto (0, 1)$ is the termination condition, which determines the probability of option termination based on the current state, previous state, and parameters.*
- *$\pi_p : S \times \theta_p \mapsto A$ is the option policy, which specifies the action to be taken in a given state and with specific parameters.*

The initiation set I_o is determined as the set of states in S that satisfy the precondition $pre(o)$ when instantiated with any valid assignment α from θ_o, denoted as $L(s) \models pre(o(\alpha))$. Before delving into the initialization of the termination condition for the parameterized options, there are a few terms that need to be explained.

We initialize a parameterized option for each schematic operator o in the following manner:

1. The parameter space θ_o is defined as the space of assignments to the parameters of schematic operator o.
2. The initiation set I_o is determined as the set of states in S that satisfy the precondition $pre(o)$ when instantiated with any valid assignment α from θ_o, denoted as $L(s) \models pre(o(\alpha))$. Mathematically expressed as, $I_o = \{s \in S | L(s) \models pre(o(\alpha)), \exists \alpha \in \theta_o\}$.

Before delving into the initialization of the termination condition for the parameterized options, there are a few terms that need to be explained.

Definition 3 *The **context** of a state s for a ground operator $o(\alpha)$ refers to the set of facts that are neither part of the precondition nor the effect of the operator. In formal terms, the context $C_o(s, \alpha)$ is defined as the set of literals in $L(s)$ that are not present in the union of the instantiated precondition $pre(o((\alpha)))$ and the instantiated effect $eff(o(\alpha))$. Mathematically, we can express it as:*

$$C_o(s, \alpha) = L(s) \setminus \{pre(o(\alpha)) \cup eff(o(\alpha))\}$$

Definition 4 *The **frame** of a state s for a ground operator $w\alpha$ refers to the set of facts that remain unchanged after executing the corresponding action. In formal terms, the frame $F_o(s, \alpha)$ is defined as the set of literals in $L(s)$ that are preserved or unaffected by the execution of the action represented by o with the parameter assignment α. This is equivalently expressed as the set of facts in the context and prevali conditions. Mathematically, we can express it as:*

$$F_o(s, \alpha) = C_o(s, \alpha) \cup prv(o(\alpha))$$

Using the above definitions of the context and frame, we can now define the termination condition of the parameterized option as follows: The termination condition $\beta_o(s, s', \alpha)$ for the parameterized option associated with schematic operator o and parameter assignment α is defined as:

$$\beta_o(s, s', \alpha) = \begin{cases} 1 \text{ if } L(s') \models eff(o(\alpha)) \\ \quad \text{and } L(s') \models F_o(s, \alpha) \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

This means that the termination condition returns 1 when the resulting state s' satisfies the effect of the operator with the given parameter assignment α, and the frame $F_o(s, \alpha)$ conditions. Otherwise, the termination condition returns a value of 0, indicating that the option should continue.

Additionally, we also define an unparameterized *goal option*, similar to Lee et al. (2022). To train the parameterized options, we define an intrinsic reward that encourages adherence to the frame conditions. This reward is formulated as a linear combination of three components. The first component is the step cost, which is a negative constant assigned for every action executed by the agent. This encourages the agent to complete the task in as few steps as possible. The second component is a conditional positive reward, which is provided only when the agent successfully terminates the parameterized option. This reward is associated with achieving the intended goal or subgoal of the option. The third component is a conditional negative constant that is given when the frame conditions of the option is violated. The frame conditions ensures that the context of the state remains unchanged after executing the option. If the context changes, indicating a violation of the frame constraint, the agent receives a negative reward. Formally, the reward $R_o(s, s', \alpha)$ for a transition (s, s') with the parameter assignment α for the parameterized option associated with schematic operator o is defined as:

$$R_o(s, s', \alpha) = \underbrace{c_1}_{\text{step cost}} + \underbrace{c_2 \mathbb{I}(\beta_o(s, s', \alpha))}_{\text{successful termination}} + \underbrace{c_3 \mathbb{I}(s' \not\models F_o(s, \alpha))}_{\text{frame violation}} \tag{2}$$

where c_1, c_2, and c_3 are constants, and $\mathbb{I}(P)$ is an indicator function that evaluates to 1 if P is `True`, and 0 otherwise. This formulation of the reward provides a learning signal for the agent to optimize its policy with respect to achieving successful termination while adhering to the frame constraint. The overall training procedure is described in Algorithm 1.

Algorithm 1: Learning Parameterized PaRL

INPUT: PaRL task distribution $\mathcal{E}$, operators $\mathcal{O}$, #tasks to train T, #episodes to train K,
OUTPUT: Parameterized options policies $\pi_o, \forall o \in \mathcal{O}$

$\pi_o, \mathcal{B}_o, \forall o \in \mathcal{O}$
▷ initialize a policy and a buffer for each operator
for task $t = 0$ to T **do**
  $\langle M, \Pi, L \rangle \sim P(\mathcal{E})$
  ▷ sample a PaRL task
  for episode $k = 0$ to K **do**
    s, G = M.reset()
    ▷ reset env and observe state and goal
    $\bar{s}, \mathcal{G} = L(s), L(G)$
    ▷ get planning state and goal
    plan = getPlan($\Pi, \hat{s}, \mathcal{G}$) + goal option
    ▷ Get a sequence of operators and assignments
    e_done, o_done = `False`, `False`
    ▷ done flags for episode and option
    for $o(\alpha)$ in plan **do**
      while ¬ e_done and ¬ o_done **do**
        $a = \pi_o(s, \alpha)$
        ▷ Sample an action from the resp. policy
        s', done = M.step(s, a) ▷ Take a step in env
        $\bar{s}' = L(s')$
        $r = c_1 + c_3 \mathbb{I}(s' \not\models F_o(s, \alpha))$
        ▷ Step cost & frame constraint
        o_done = $\mathbb{I}(\beta_o(s, s', \alpha))$
        if o_done **then**
          $r = r + c_2$
        end if
        $\mathcal{B}_o = \mathcal{B}_o \cup \langle s || \alpha, a, r, s' || \alpha \rangle$
        ▷ Adding experience in the buffer
        $s = s'$
      end while
    end for
  end for
  for all $o \in \mathcal{O}$ **do**
    π_o = update($\pi_o, \mathcal{B}_o$)
    ▷ Update the policy using off-policy algorithm
  end for
end for
return $\pi_o, \forall o \in \mathcal{O}$

4 Experiments

Our initial experiments are designed to address the following research questions:

Q1. Is learning parameterized policies sample efficient?

Q2. Do the parameterized policies demonstrate improved generalization abilities?

The answers to these questions will provide valuable insights into the effectiveness and potential advantages of employing parameterized policies in our approach.

To investigate and answer our research questions, we have defined a PaRL task distribution within the mini-grid domain. The mini-grid domain represents a grid world consisting of multiple interconnected rooms. Figure 2 illustrates a few examples of the mini-grid environment. Each room can have locked or unlocked doors, with each door having a specific color. Keys of corresponding colors can be used to lock or unlock the doors, and these keys may be located in different rooms. The objective in the mini-grid domain is to navigate an agent to a designated goal location. The goal location is represented by a green indicator, while the agent is depicted as a red triangle in Figure 2. Within this domain, different PaRL tasks can be defined, encompassing various configurations and attributes. These variations may include doors at different locations, doors and keys of different colors and quantities, distinct initial and goal room assignments, varying numbers of rooms, balls, or other elements. Moreover, within a specific PaRL task, different episodes may present diverse initial and goal locations within the same room.

In our implementation, we utilize Ray RLlib (Liang et al. 2017) as the framework for implementing our approach. Specifically, for learning the base policy, we employ the Proximal Policy Optimization (PPO) algorithm, which is implemented within Ray RLlib. To facilitate the learning process, we utilize the same planning domain description as presented in Lee et al. (2022). We adopt an egocentric image representation for the mini-grid. An example transformation is illustrated in Figure 4, which is an egocentric transformation of state shown in Figure 3a. In this representation, the agent is always positioned at the bottom center of the image and facing upwards. The egocentric image representation consists of six channels: doors, walls, balls, agent, keys, and goal. Different objects are represented by distinct values in the respective channels, allowing the agent to perceive the environment from an egocentric viewpoint. For the parameterized options, we extend the image representation by incorporating additional channels that represent the parameter assignments. These parameter assignments are concatenated with the original image representation, forming a comprehensive input representation for the parameterized options. By leveraging Ray RLlib, incorporating the egocentric image representation, and representing parameter assignments as additional channels, our implementation provides a robust framework for training and utilizing parameterized policies.

To address the research question Q1 regarding the sample efficiency of learning parameterized options, we conducted a comparative analysis with two prior baselines: standard Deep RL with the PPO algorithm and PaRL with PPO as the base learner. It is evident from the curves, shown in Fig 5, that our approach of learning parameterized options outperforms both the PaRL and RL baselines in terms of sample efficiency. This finding directly answers question Q1 by demonstrating that parameterized options can indeed be learned in a sample-efficient manner, even in the presence of a larger state space.

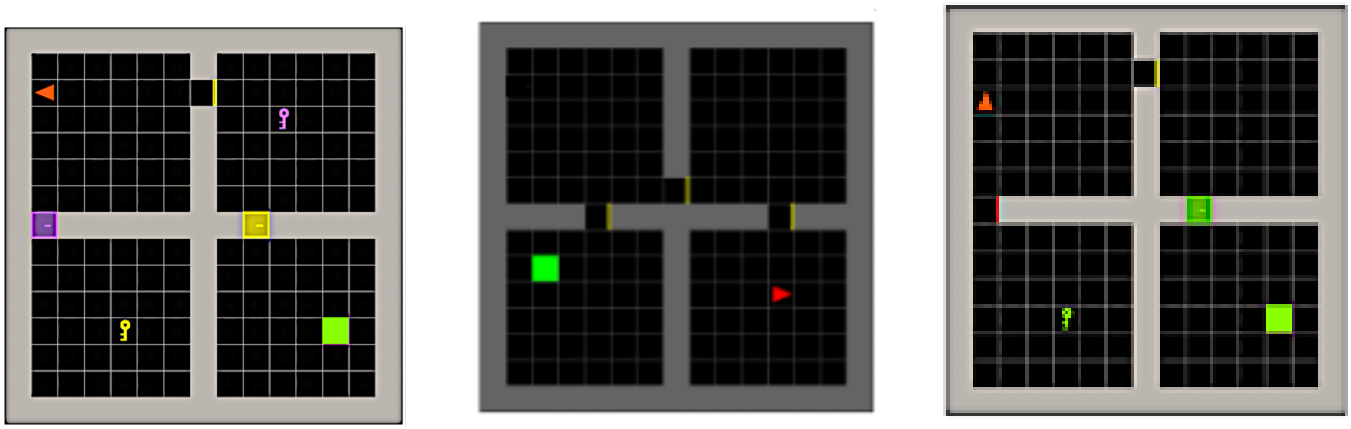
To address question Q2, we conducted evaluations on additional PaRL tasks using the policies trained on the training task (shown in Figure 3a). The evaluation tasks consisted of three variations: (1) randomly sampling the room for the initial location of the agent, (2) randomly sampling the room for the goal location, and (3) randomly sampling rooms for both the initial and goal locations. The performance of the agents on these evaluation tasks is depicted in Figure 6. The results of the evaluations demonstrate that while PaRL exhibits some level of generalization to these random evaluation tasks, our approach of learning parameterized policies showcases significantly improved generalization efficiency.

Further, we conducted evaluations on MiniGrid PaRL tasks with randomly sampled colors for the doors and keys. The objective was to examine how well the policies could adapt to variations in key colors. The success rates of the evaluations are compared and illustrated in Figure 7. The results clearly demonstrate that our approach, utilizing parameterized options, exhibits a similar success rate in generalizing to different key colors as it does in generalizing to random initial and goal locations. In contrast, the PaRL baseline struggles to generalize effectively in the face of changing key colors. The lower success rate observed in the PaRL approach highlights its limitations in handling variations that are specific to objects, such as different key colors.

In summary, our evaluation results provide strong evidence that learning parameterized options yields superior generalization capabilities when dealing with variations related to objects. This highlights the advantages of our approach and reinforces the significance of utilizing parameterized policies for enhanced generalization in reinforcement learning tasks involving objects.

5 Related work

Various promising approaches have recently emerged to learn generalizable policies. Schaul et al. (2015) proposed a framework for learning parameterized value functions known as Universal Value Function Approximators (UVFAs). This approach extends traditional value functions by incorporating both states and goals as inputs, allowing for more flexible and generalizable value estimation. The key idea behind UVFAs is to factorize the value function into two components: a goal representation and a state representation. By separately encoding the goal and state information, UVFAs can capture the value of being in a particular state while aiming to achieve a specific goal. da Silva, Konidaris, and Barto (2012) propose a method for learning reusable motor skills with parameters. The method focuses on acquiring skills that can be adapted and applied to different tasks or situations by adjusting their parameters. While UVFAs and parameterized skills focus on generalizing policies across tasks or goals, our work introduces a novel per-



(a) Training task (b) Different initial and goal room (c) Different Key Color

Figure 3: Three PaRL tasks used in evaluation

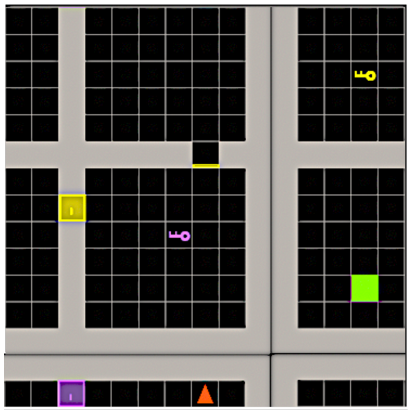


Figure 4: Egocentric representation of state in Fig. 3a

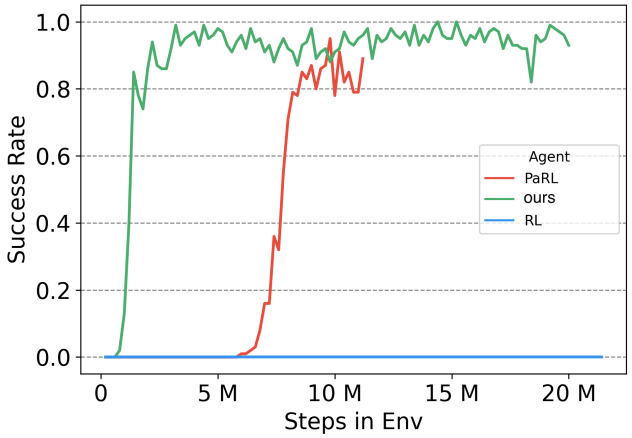


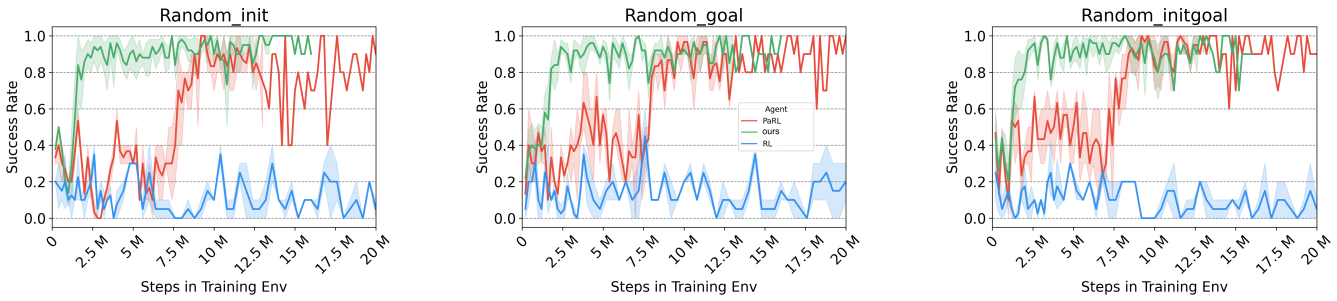
Figure 5: Learning curves of different agents on training task (shown in Fig 3a)

spective by parameterizing options to represent object assignment. This differs from both universal value functions (UVFAs) and parameterized skills approaches, where the parameters represent goals or tasks, respectively.

In a recent work by Kokel et al. (2022a), they proposed the RePReL framework, which integrates planning and reinforcement learning. Our proposed framework shares similarities with the RePReL framework in that both frameworks define a single option for each schematic operator. However, there is a notable difference in terms of the additional influence information required in the RePReL framework to generate abstract representations of the state and differentiate among the ground operators. The RePReL framework employs first-order conditional influence information to generate abstract state representations. These representations effectively capture the contextual aspects required for the options to acquire context-specific behaviors. In contrast, our approach learns parameterized policies for options.

6 Conclusion

In conclusion, this paper presents a novel approach for defining parameterized options using operator descriptions in the context of hierarchical reinforcement learning. The study focuses on utilizing AI planning techniques to establish options for acquiring primitive or low-level skills. The empirical evaluations conducted on the mini-grid domain demonstrate the effectiveness of the proposed approach. The results indicate that the approach enhances sample efficiency during training, allowing for more efficient exploration and learning. The approach overcomes limitations associated with generalization to problems with different objects. The findings of this study contribute to the growing body of research that combines AI planning and hierarchical reinforcement learning. The proposed approach provides a valuable technique for defining parameterized options, expanding the possibilities for more efficient and adaptable learning in complex environments.



(a) Randomly sampled initial room

(b) Randomly sampled goal room

(c) Randomly sampled initial as well as goal room

Figure 6: Different evaluation tasks

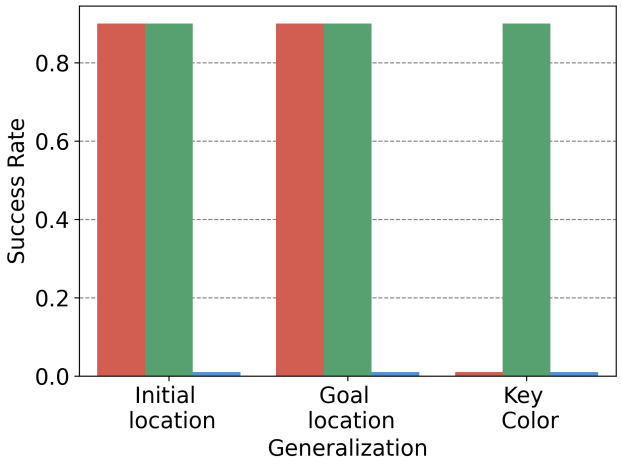


Figure 7: Comparison of the generalization ability

References

da Silva, B. C.; Konidaris, G. D.; and Barto, A. G. 2012. Learning Parameterized Skills. In *ICML*.

Dietterich, T. G. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In *ICML*, 118–126.

Eppe, M.; Nguyen, P. D. H.; and Wermter, S. 2019. From Semantics to Execution: Integrating Action Planning With Reinforcement Learning for Robotic Causal Problem-Solving. *Frontiers in Robotics and AI*.

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic Plans as High-Level Instructions for Reinforcement Learning. *ICAPS*, 540–550.

Jin, M.; Ma, Z.; Jin, K.; Zhuo, H. H.; Chen, C.; and Yu, C. 2022. Creativity of AI: Automatic Symbolic Option Discovery for Facilitating Deep Reinforcement Learning. In *AAAI*, 7042–7050. AAAI Press.

Kokel, H.; Manoharan, A.; Natarajan, S.; Ravindran, B.; and Tadepalli, P. 2021. RePReL: Integrating Relational Planning and Reinforcement Learning for Effective Abstraction. *ICAPS*, 31(1): 533–541.

Kokel, H.; Natarajan, S.; Ravindran, B.; and Tadepalli, P. 2022a. RePReL: A Unified Framework for Integrating Relational Planning and Reinforcement Learning for Effective Abstraction in Discrete and Continuous Domains. *Neural Computing and Applications*.

Kokel, H.; Prabhakar, N.; Ravindran, B.; Blasch, E.; Tadepalli, P.; and Natarajan, S. 2022b. Hybrid Deep RePReL: Integrating Relational Planning and Reinforcement Learning for Information Fusion. In *FUSION*, 1–8. IEEE.

Lee, J.; Katz, M.; Agravante, D. J.; Liu, M.; Tasse, G. N.; Klinger, T.; and Sohrabi, S. 2022. Hierarchical Reinforcement Learning with AI Planning Models.

Liang, E.; Liaw, R.; Nishihara, R.; Moritz, P.; Fox, R.; Gonzalez, J.; Goldberg, K.; and Stoica, I. 2017. Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*, 85.

Lyu, D.; Yang, F.; Liu, B.; and Gustafson, S. 2019. Sdrl: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *AAAI*.

McDermott, D. V. 2000. The 1998 AI Planning Systems Competition. *AI Mag.*, 21(2): 35–55.

Parr, R.; and Russell, S. J. 1998. Reinforcement learning with hierarchies of machines. In *NeurIPS*.

Sarathy, V.; Kasenberg, D.; Goel, S.; Sinapov, J.; and Scheutz, M. 2021. SPOTTER: Extending Symbolic Planning Operators through Targeted Reinforcement Learning. In *AAMAS*, 1118–1126. ACM.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal Value Function Approximators. In *ICML*.

Silver, T.; Athalye, A.; Tenenbaum, J. B.; Lozano-Pérez, T.; and Kaelbling, L. P. 2022. Learning Neuro-Symbolic Skills for Bilevel Planning. In *CoRL*, volume 205 of *Proceedings of Machine Learning Research*, 701–714. PMLR.

Sutton, R. S.; Precup, D.; and Singh, S. 1998. Intra-Option Learning about Temporally Abstract Actions. In *ICML*, 556–564. Morgan Kaufmann.