AI Planning Annotation in Reinforcement Learning: Options and Beyond

Junkyu Lee, Michael Katz, Don Joven Agravante, Miao Liu,
Tim Klinger, Murray Campbell, Shirin Sohrabi, Gerald Tesauro
IBM Research AI
{Junkyu.Lee,michael.katz1,Don.Joven.R.Agravante,miao.liu1}@ibm.com, {tklinger,mcam,ssohrab,gtesauro}@us.ibm.com

Abstract

AI planning and reinforcement learning (RL) both solve sequential decision-making problems, taking fundamentally different approaches. In this work, we aim to bring AI planning and RL closer by investigating the relationship between abstractions in AI planning and the options framework in RL. To this end, we propose annotating RL tasks with AI planning models, allowing us to define options based purely on the planning model. Our experimental investigation shows that these options can be quickly trained offline and can improve the sample efficiency of a reinforcement learning algorithm.

Introduction

While both AI planning and reinforcement learning (RL) solve sequential decision-making problems, the approach they take is quite different. AI planning finds goal directed paths in large scale state transition systems, concisely symbolically represented using a logic-base language, such as PDDL (McDermott 2000), and is able to quickly solve large tasks. Deep reinforcement learning (RL) approaches directly collect samples from a target environment and do not require predefined symbolic models for solving problem tasks, except for the assumption that the underlying model is a Markov decision process (MDP). However, such model-free RL approaches are notoriously known for their sample inefficiency, especially for large state or action space or sparse reward tasks. This limits a lot of the real-world applications (Dulac-Arnold, Mankowitz, and Hester 2019).

The differences in AI planning and RL might make it seem like they are at odds with each other. However, there are several similarities and complementary pieces that can be exploited to address the shortcomings of the other's approach. In fact, this observation is not new and several papers study the close relationship between the two fields (e.g., (Ryan 2002; Grounds and Kudenko 2007; Leonetti, Iocchi, and Stone 2016; Eysenbach, Salakhutdinov, and Levine 2019)). In this paper, we aim to improve the sample efficiency of RL by using parallels between the hierarchical reinforcement learning formalism and abstractions in AI planning.

Hierarchical reinforcement learning (Barto and Mahadevan 2003) aims to address the curse of dimensionality by exploiting the task structure in the state and action space. Earlier works introduced manually designed task decomposition (Dietterich 2000) but recent works have moved toward learning or automatically discovering such structure. The options framework (Sutton, Precup, and Singh 1999) is often used as a basis for this structure. These *options* define temporal abstractions that induce a semi-Markov decision process (SMDP) over the underlying MDP. From this, several works have proposed to discover options (Machado, Bellemare, and Bowling 2017; Bacon, Harb, and Precup 2017). In contrast to fully specified decompositions, (Bacon, Harb, and Precup 2017) proposes an end-to-end learning of the structure. In between these extremes, we propose a middle ground by using planning abstractions and symbolic annotation as a way to define options. In this setting, our work is closest to the work of taskable RL (Illanes et al. 2020) and we differ in the symbolic annotation requirements. A key insight is that options, which are usually thought of as temporal abstractions, have a general formalism that allows the use of state abstractions which are well-studied in the AI planning literature (Edelkamp 2001; Hoffmann, Sabharwal, and Domshlak 2006; Helmert et al. 2014; Katz and Domshlak 2010; Torralba and Sievers 2019).

In this paper, we present an integrated framework for planning and RL by linking the state abstraction in planning and temporal abstraction in RL. We first formalize the framework and investigate the options discovered by state abstraction based on projection. Then, we demonstrate that an RL algorithm that learns option level policy functions in SMDP setting combined with proximal policy optimization algorithm (PPO) (Schulman et al. 2017) and show the improvements in the sample efficiency in the room navigation domain and logistics domain in AI planning.

Background

RL and Options Framework

In reinforcement learning, an agent interacts with an environment which we formulate as an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$ with a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a state transition function $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discounting factor $\gamma \in (0, 1]$ for the rewards. The most common objective is to learn a stationary optimal policy π^* that maximizes the expected

return,

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\Big], \quad (1)$$

where s_0 is the initial state, and $\pi(a|s)$ is a probability distribution $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Given a policy π, a value function $V^{\pi}(s)$ is the expected sum of discounted reward from each state $s \in \mathcal{S}$,

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\Big[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)V^{\pi}(s')\Big]. \quad (2)$$

The value of executing an action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ under the policy π is important in learning algorithms and we call it an action-value function,

$$Q^{\pi}(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)Q^{\pi}(s'). \quad (3)$$

The optimal value function $V^*(s)$ and action-value function $Q^*(s, a)$ can be found by maximizing $V^{\pi}(s)$ and $Q^{\pi}(s, a)$,

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad (4)$$

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a). \quad (5)$$

A set of options $\mathcal{O}$ formalizes the temporally extended actions that defines a semi-MDP (SMDP) over the original MDP $\mathcal{M}$. A Markovian option $O \in \mathcal{O}$ is a triple $\langle \mathcal{I}_O, \pi_O, \beta_O \rangle$, where $\mathcal{I}_O$ is the initiation set $\mathcal{I}_O \subseteq \mathcal{S}$ that defines the states in which O can begin, π_O is a stationary intra-option policy that provides mapping from a state to an action $\pi_O : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and β_O is a termination function $\beta_O : \mathcal{S} \to [0, 1]$ that defines the probability of terminating the O in state s. We follow the call-and-return option execution model, where an agent selects an option O using an option level policy $\mu(O|s)$ in state s at the time t, and generates a sequence of actions according to the intra-policy $\pi_o(a|s)$. The execution of the option O continues up to k time steps and terminated by β_O and returns option reward accumulated from time $t + 1$ to $t + k$ with a discounting factor γ,

$$R(s, O) = \mathbb{E}\Big[\sum_{t'=t+1}^{t+k} \gamma^{t'-t-1} r_{t'} | \mathcal{E}(O, s, t)\Big], \quad (6)$$

where $\mathcal{E}(O, s, t)$ denotes the event of an option O being selected in state s at time t, and $r_{t'}$ denotes the reward obtained at time t'. The state transition under the options can be written as

$$P(s'|s, o) = \sum_{j=0}^{\infty} \gamma^j Pr\Big(k = j, s_{t+j} | \mathcal{E}(O, s, t)\Big), \quad (7)$$

where the probability is a combination of an option O that started at time t terminates in j steps (Sutton, Precup, and Singh 1999).

In SMDP theory, the value function $V^{\mu}(s)$ under the option level policy μ and the option-value function $Q^{\mu}(s, O)$ can be derived in a functional form similar to Eq. (2) and Eq. (3). Namely,

$$V^{\mu}(s) = \sum_{O \in \mathcal{O}} \mu(O|s)\Big[R(s, O) + \sum_{s' \in \mathcal{S}} P(s'|s, O)V^{\mu}(s')\Big], \quad (8)$$

and the option-value function $Q^{\mu}(s, O)$,

$$Q^{\mu}(s, O) = R(s, O) + \sum_{s' \in \mathcal{S}} P(s'|s, O) \sum_{O \in \mathcal{O}} \mu(O|s)Q^{\mu}(s', O). \quad (9)$$

Given a set of learned options $\mathcal{O}$, off-policy learning methods such as Q-learning (Watkins and Dayan 1992) and Deep Q-learning (Mnih et al. 2013) can learn the option value function Eq. (9) by SMDP Q-learning or intra-option Q-learning (Sutton, Precup, and Singh 1999). On the other hand, policy optimization methods such as actor-critic (Konda and Tsitsiklis 2000) and proximal policy optimization algorithm (Schulman et al. 2017) can directly optimize the expected return in Eq. (8) to obtain the option level policy μ by extending the set of learned options $\mathcal{O}$ with primitive actions $\mathcal{A}$.

Planning

In this work we follow the notation of (Bäckström and Nebel 1995), omitting the initial state from the task definition. SAS$^+$ planning task Π is given by a tuple $\langle \mathcal{V}, \mathcal{O}, s_* \rangle$, where $\mathcal{V}$ is a set of *state variables* and $\mathcal{O}$ is a finite set of *operators*. Each state variable $v \in \mathcal{V}$ has a finite domain $dom(v)$ of values. A pair $\langle v, \vartheta \rangle$ with $v \in \mathcal{V}$ and $\vartheta \in dom(v)$ is called a *fact*. A (partial) assignment to $\mathcal{V}$ is called a *(partial) state*, with the partial state s_* being the *goal*. We denote the variables of a partial assignment p by $\mathcal{V}(p)$. It is convenient to view a partial state p as a set of facts with $\langle v, \vartheta \rangle \in p$ if and only if $p[v] = \vartheta$. Partial state p is *consistent* with state s if $p \subseteq s$. We denote the set of states of a planning task by $\mathcal{S}'$. Each *operator* $\mathcal{O}$ is a pair $\langle pre(o), \mathit{eff}(o) \rangle$ of partial states called *preconditions* and *effects*. The (possibly empty) subset of operator precondition that does not involve variables from the effect is called *prevail condition*, $prv(o) = \{\langle v, \vartheta \rangle \mid \langle v, \vartheta \rangle \in pre(o), v \notin \mathcal{V}(\mathit{eff}(o))\}$. An *operator cost* is a mapping $C : \mathcal{O} \to \mathbb{R}^{0+}$. An operator o is applicable in a state $s \in \mathcal{S}'$ if and only if $pre(o)$ is consistent with s. Applying o changes the value of v to $\mathit{eff}(o)[v]$, if defined. The resulting state is denoted by $s[\![o]\!]$. An operator sequence $\pi = \langle o_1, \ldots, o_k \rangle$ is applicable in s if there exist states $s_0, \cdots, s_k$ such that (i) $s_0 = s$, and (ii) for each $1 \leq i \leq k$, o_i is applicable in s_{i-1} and $s_i = s_{i-1}[\![o_i]\!]$. We denote the state s_k by $s[\![\pi]\!]$. π is a plan for s iff π is applicable in s and s_* is consistent with $s[\![\pi]\!]$.

A transition graph of the planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_* \rangle$ is a tuple $\mathcal{T} = \langle \mathcal{S}, T, S_* \rangle$, where $\mathcal{S}$ is the set of states of Π, $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{O} \times \mathcal{S}$ is a set of (labelled) transitions, and $\mathcal{S}_* \subseteq \mathcal{S}$ is the set of goal states. An abstraction of the transition graph $\mathcal{T}$ is a pair $\langle \mathcal{T}', \alpha \rangle$, where $\mathcal{T}' = \langle \mathcal{S}', T', S'_* \rangle$ is an *abstract transition graph* and $\alpha : \mathcal{S} \mapsto \mathcal{S}'$ is an *abstraction mapping*, such that $\langle \alpha(s), o, \alpha(s') \rangle \in T'$ for all $\langle s, o, s' \rangle \in T$, and $\alpha(s) \in \mathcal{S}'_*$ for all $s \in \mathcal{S}_*$. If $\mathcal{T}'$ does not contain any additional transitions or goal states (iff in the above definition), it is called a *homomorphism*. A homomorphism defined by a mapping α such that $\alpha(s) = \alpha(s')$ iff $s[v] = s'[v]$ for all variables $v \in \mathcal{V}'$ is called a *projection* onto variables $\mathcal{V}' \subseteq \mathcal{V}$.

Annotating RL with Planning

A planning annotated reinforcement learning (PaRL) task is defined by a triplet $E = \langle \mathcal{M}, \Pi, L \rangle$, where $\mathcal{M}$ is an MDP over states $\mathcal{S}$, Π is a planning task over states $\mathcal{S}'$, and $L : \mathcal{S} \mapsto \mathcal{S}'$ is a mapping from the states of the MDP to planning states. The objective of the PaRL task is to find an optimal policy for the MDP M. Therefore, the planning task and the mapping L can be viewed as an additional information, not essential for solving the task.

Note that additional PaRL tasks can be obtained from $E = \langle \mathcal{M}, \Pi, L \rangle$ by abstracting the planning task. Given an abstraction mapping α of the planning task Π, let Π^α be a planning task with the transition system being the abstraction under α of the transition system of Π, and $L^\alpha = \alpha \circ L$ be the function composition of α and L. Then $E' = \langle \mathcal{M}, \Pi^\alpha, L^\alpha \rangle$ is also a PaRL task, annotating the same MDP with a different planning task. While in general it is not clear how to create a concisely represented planning task Π^α for any abstraction mapping α, many existing abstraction methods allow for such concise representation to be automatically obtained (Edelkamp 2001; Hoffmann, Sabharwal, and Domshlak 2006; Katz and Domshlak 2010; Torralba and Sievers 2019). While the benefit of further abstracting the planning task might not be obvious, it will be explained in what follows.

One question worth asking is - where do the planning tasks that annotate the existing MDPs come from? In this work we assume that the planning tasks, as well as the state mappings are provided together with the MDP. Obtaining the planning task and the mapping is outside the scope of our paper and is investigated under the umbrella of work on learning the planning tasks, e.g., (Aineto, Jiménez, and Onaindia 2018).

Planning, Abstractions, Options and RL

The generic definition of planning annotated reinforcement learning tasks is a mixed blessing. On the one hand, it does not pose any constraints on the connection between the RL task and the planning task beyond the existence of the mapping L. On the other hand, if the two tasks are unrelated, it is not clear what is the benefit of connecting these tasks together. We formulate the connection between the two tasks by extending the definition of abstraction to PaRL tasks.

Definition 1 *Let $E = \langle \mathcal{M}, \Pi, L \rangle$ be a PaRL task and $\mathcal{T} = \langle \mathcal{S}', T, S_* \rangle$ be the transition graph of Π. We say that $\langle \Pi, L \rangle$ is an abstraction of $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$ if for all $\langle s, a, s' \rangle$ we have $P(s'|s, a) > 0$, iff $\langle L(s), o, L(s') \rangle \in T$ for some $o \in \mathcal{O}$ or $L(s) = L(s')$. We call such PaRL tasks properly planning annotated reinforcement learning (PPaRL) tasks.*

There are two major differences from the original definition of transition graph abstractions. First, P is not necessarily deterministic. Second, the definition does not require mapping goal states (not defined for MDPs) to goal states. However, the definition does allow us to reflect the abstraction of the transition graph *structure*.

For a PPaRL task $E = \langle \mathcal{M}, \Pi, L \rangle$, each operator o of Π defines a sub-MDP $\mathcal{M}_o$ by requiring a non-zero transition probability to be only on $\langle s, a, s' \rangle$ where either $\langle L(s), o, L(s') \rangle \in T$ or $L(s) = L(s')$.

RL Options Definition

Given a PaRL task $E = \langle \mathcal{M}, \Pi, L \rangle$, we define the set of options $\mathcal{O}$ for $\mathcal{M}$ as follow. For each planning task operator o, we define an option O_o by specifying analytically its initiation set and its termination function.

$$\mathcal{I}_{O_o} = \{ s \in \mathcal{S} \mid \mathit{pre}(o) \subseteq L(s) \}$$

In words, a state $s \in \mathcal{S}$ of $\mathcal{M}$ is in the initiation set of the option O_o if the operator o is applicable in the corresponding planning state $L(s)$.

$$\beta_{O_o}(s) = \begin{cases} 1 & (\mathit{prv}(o) \cup \mathit{eff}(o)) \subseteq L(s), \\ 0 & \text{otherwise.} \end{cases}$$

In words, the termination function of the option O_o maps a state $s \in \mathcal{S}$ of $\mathcal{M}$ to 1 if the operator o can result in the corresponding abstract state $L(s)$. Otherwise, it maps s to 0.

Additionally, for a goal s_* and an operator o, if the operator can achieve the goal (that is, it can be the last operator of a plan), we can define an option $O_o^{s_*}$ by specifying analytically its initiation set.

$$\mathcal{I}_{O_o^{s_*}} = \{ s \in \mathcal{S} \mid \mathit{pre}(o) \cup (s_* \setminus \mathit{eff}(o)) \subseteq L(s) \}$$

In words, a state $s \in \mathcal{S}$ of $\mathcal{M}$ is in the initiation set of the option $O_o^{s_*}$ if the operator o is applicable in the corresponding abstract state $L(s)$ and applying the operator o in $L(s)$ would result in an abstract goal state. The termination function $\beta_{O_o^{s_*}}(s) = 1$ for all goal states in $\mathcal{M}$, $s \in \mathcal{S}_*$ and 0 for all other states. To complete the option definition, we need the option policy π_O that moves the agent from any state in the initiation set towards a state in the termination set. This is essentially solving a (small) portion of the full MDP. We detail this in the next subsection.

As the number of options is proportional to the number of operators in Π, it can sometimes be beneficial to abstract the planning task in a way that reduces the number of operators. One possible way to do so is *label reduction* (Sievers, Wehrle, and Helmert 2014).

Previous attempts in the literature have suggested using planning operators to define options (Illanes et al. 2020). However, they assume an additional domain specific function α is provided, associating planning operators with conditions over propositional variables. Here, we do not require such an additional input, relying solely on the planning task. Further, while previous work defined the termination function only based on the provided function α, having the initiation set consisting of all states, here we define both the initiation set and termination function based on operators preconditions and effects, respectively.

Solving PaRL

Given a PaRL task $E = \langle \mathcal{M}, \Pi, L \rangle$ and any pair of initial state $s_0 \in \mathcal{S}$ and a goal $s_g \in \mathcal{S}_*$ in $\mathcal{M}$, we can generate a sequence of options $\{O_{o_1}, O_{o_2}, \ldots, O_{o_k}\}$ from a plan in Π that reaches the goal state $L(s_g) \in \mathcal{S}'$ from the initial state $L(s_0) \in \mathcal{S}'$. Therefore, AI planners can be invoked in two

ways, either precompute those option-level plans *offline* or generate plans while training RL agents *online*. At the same time, RL agents also have the freedom to train both option level policy μ and intra-option policies π_O for all $O \in \mathcal{O}$ using either off-policy or on-policy learning algorithms.

On the one hand, we can view an intra-option Q-learning algorithm proposed in taskable RL (Illanes et al. 2020) as an online planning off-policy learning algorithm, where the sequence of options generated directly from a planner while updating π_O. As another example, algorithms based on the option critic architecture (Bacon, Harb, and Precup 2017; Harb et al. 2018; Riemer, Liu, and Tesauro 2018) can be viewed as another online planning off-policy learning approach, where the plans over the options are sampled from option level policy μ. We can design generic *online* planning strategies for training RL agents by integrating AI planning algorithms inside the rollout process in either on-policy or off-policy learning algorithms.

On the other hand, we see that option discovery algorithms relying on capturing the properties inherent in the state space often separate the option discovery, intra-policy learning, and option-level policy learning phase (Stolle and Precup 2002; Kazemitabar and Beigy 2008; Machado, Bellemare, and Bowling 2017; Ramesh, Tomar, and Ravindran 2019). In such cases, we can generate option level plans *offline* to help the learning process for identifying relevant options for solving a particular problem instance or learning option-level policy similar to skill-chaining (Konidaris and Barto 2009). In addition, PaRL could possibly improve the data generation process of offline RL.

Offline Planning with SMDP Learning

Among many possible combinations, we demonstrate our PaRL framework with an offline option planning with SMDP learning that uses PPO for policy updates. Proximal Policy Optimization (PPO) (Schulman et al. 2017) is a policy gradient method for online learning, and it uses first-order optimization and clipped probability ratio to efficiently update policies from the data. This was chosen for its reliable performance in many RL tasks.

Algorithm 1 outlines the overall procedure for training option level policy μ and subset of intra-options $\mathcal{O}$ relevant to a given PaRL task $E\langle \mathcal{M}, \Pi, L \rangle$.

SMDP learning with PPO begins by selecting the subset of relevant options by generating a plan in the annotated planning task Π given an initial state s_0 and a goal state s_g in $\mathcal{M}$ (line 1). The subset of options $\mathcal{O}_{\mathbf{op}}$ can be derived from each operator in $\mathbf{op}$ following the RL options definition in the previous section (line 2). After identifying a subset of options to train, we use PPO agent to train each $\pi_{O_{o_i}}$ subject to its initiation set $\mathcal{I}_{O_{o_i}}$, and $\beta_{O_{o_i}}$ using the sub-MDP $\mathcal{M}_{O_{o_i}}$ (line 3-4). The last stage is training μ over the SMDP defined on the original $\mathcal{M}$ with additional options $\mathcal{O}_{\mathbf{op}}$ following Eq. (8), where the SMDP extends its set of actions from the primitive actions $\mathcal{A}$ with $\mathcal{O}_{\mathbf{op}}$ (line 5-6).

In the experiment, we ran PPO implemented in Stable baselines3 (Raffin et al. 2019) with the following hyper parameters: the maximum episode length was set to 1024, the discounting factor γ was 0.99, the batch size was 64, the number of gradient updates per single rollout was 10, the λ for generalized advantage estimation was 0.95, the clip range was 0.2, the coefficient for both the entropy and value loss was 0.5, and the learning rate was 0.0003.

Algorithm 1 SMDP Learning with PPO

Require: PaRL $E\langle \mathcal{M}, \Pi, L \rangle$, the initial state $s_0 \in \mathcal{S}$ and goal state $s_g \in \mathcal{S}_*$ in $\mathcal{M}$
Ensure: option level policy μ, trained options $\mathcal{O}$

Offline Planning for Option Selection
1: $\mathbf{op} \leftarrow$ generate a plan in Π from $L(s_0)$ to $L(s_g)$
2: $\mathcal{O}_{\mathbf{op}} \leftarrow \{O_{o_i} \in \mathcal{O} | o_i \in \mathbf{op}\}$
Intra-policy Learning
3: **for** each option $O_{o_i} \in \mathcal{O}_{\mathbf{op}}$ **do**
4: $\quad \pi_{O_{o_i}} \leftarrow \text{PPO}(\mathcal{M}_{O_{o_i}})$ ▷ train $\pi_{O_{o_i}}$ s.t. $\mathcal{I}_{O_{o_i}}, \beta_{O_{o_i}}$
Option-level SMDP Learning
5: $\tilde{A} \leftarrow \mathcal{A} \cup \mathcal{O}_{\mathbf{op}}$
6: $\mu \leftarrow \text{PPO}(\mathcal{M}' \langle \mathcal{S}, \tilde{A}, P, r, \gamma \rangle)$

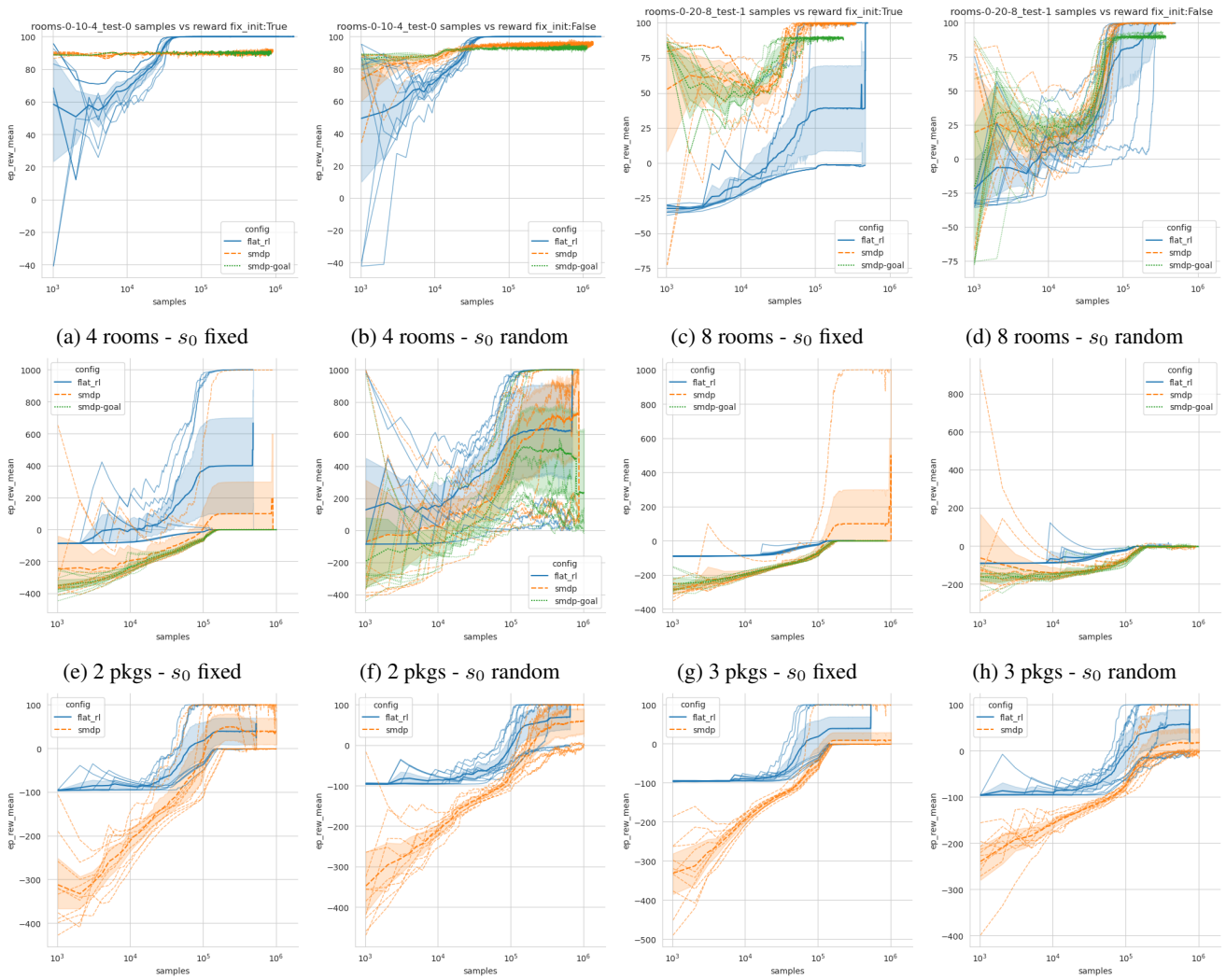
Experimental Evaluation

We implemented our framework on top of PDDLGym (Silver and Chitnis 2020), Stable baselines3 (Raffin et al. 2019), and option-critic baselines (Zhang 2018). In order to empirically evaluate our proposed approach, we obtain a PPaRL task $E = \langle \mathcal{M}, \Pi, L \rangle$ as follows. We start with a planning task Π_M as an RL problem and obtain an equivalent MDP $\mathcal{M}$ by using PDDLGym. Then, we perform a variable projection mapping (Helmert, Haslum, and Hoffmann 2007) onto the goal variables of Π_M and obtain a planning task Π and a mapping α from the states of Π_M to the states of Π. Together with the mapping L_M from the states of $\mathcal{M}$ to the states of Π_M provided by PDDLGym, this allows us to define $L = \alpha \circ L_M$. We evaluated MDP tasks that are obtained from the `4-rooms` domain (Sutton, Precup, and Singh 1999) and the `blocksworld` and `logistics` domains in classical planning. We compare the following three configurations for training RL agents and report the *average rewards* and the *average lengths of the plans* during training. The baseline configuration, `flat-rl`, learns the policy function from the flat MDP with Proximal Policy Optimization (PPO) (Schulman et al. 2017), and the next configuration, `smdp`, implements Algorithm 1. Lastly, we also evaluated the `option-critic` algorithm, (Bacon, Harb, and Precup 2017) for training both option level policy function and a set of intra-policy functions as a baseline for comparing the performance.

Rooms Domain

The rooms domain modifies the classical 4-rooms domain by increasing the number of rooms in the varying size of the grids. In this problem, an agent moves on a grid, separated into N rooms, with narrow corridors connecting the adjacent rooms. The agent needs to move from a given location on the grid (in a given room) to a goal location (in a goal room).

In order to use the domain within our framework, we have created a PDDL version of the N rooms domain and defined



(a) 4 rooms - s_0 fixed (b) 4 rooms - s_0 random (c) 8 rooms - s_0 fixed (d) 8 rooms - s_0 random

(e) 2 pkgs - s_0 fixed (f) 2 pkgs - s_0 random (g) 3 pkgs - s_0 fixed (h) 3 pkgs - s_0 random

(i) 4 blocks - s_0 fixed (j) 4 blocks - s_0 random (k) 4 blocks - s_0 fixed (l) 4 blocks - s_0 random

Figure 1: Learning curves showing the rewards over the 10^6 samples. The first row presents the results from the `rooms` domain, the second row presents the results from the `logistics` domain, and the third row presents the results from the `blocksworld` domain. Each plot shows 10 independent runs and the aggregated results with the 95% confidence interval. The blue lines are the configuration `flat-rl`, The orange lines are the configuration `smdp` without using the goal options $\mathcal{I}_{O_o^{s*}}$, and the green lines are the configuration `smdp` using the goal options.

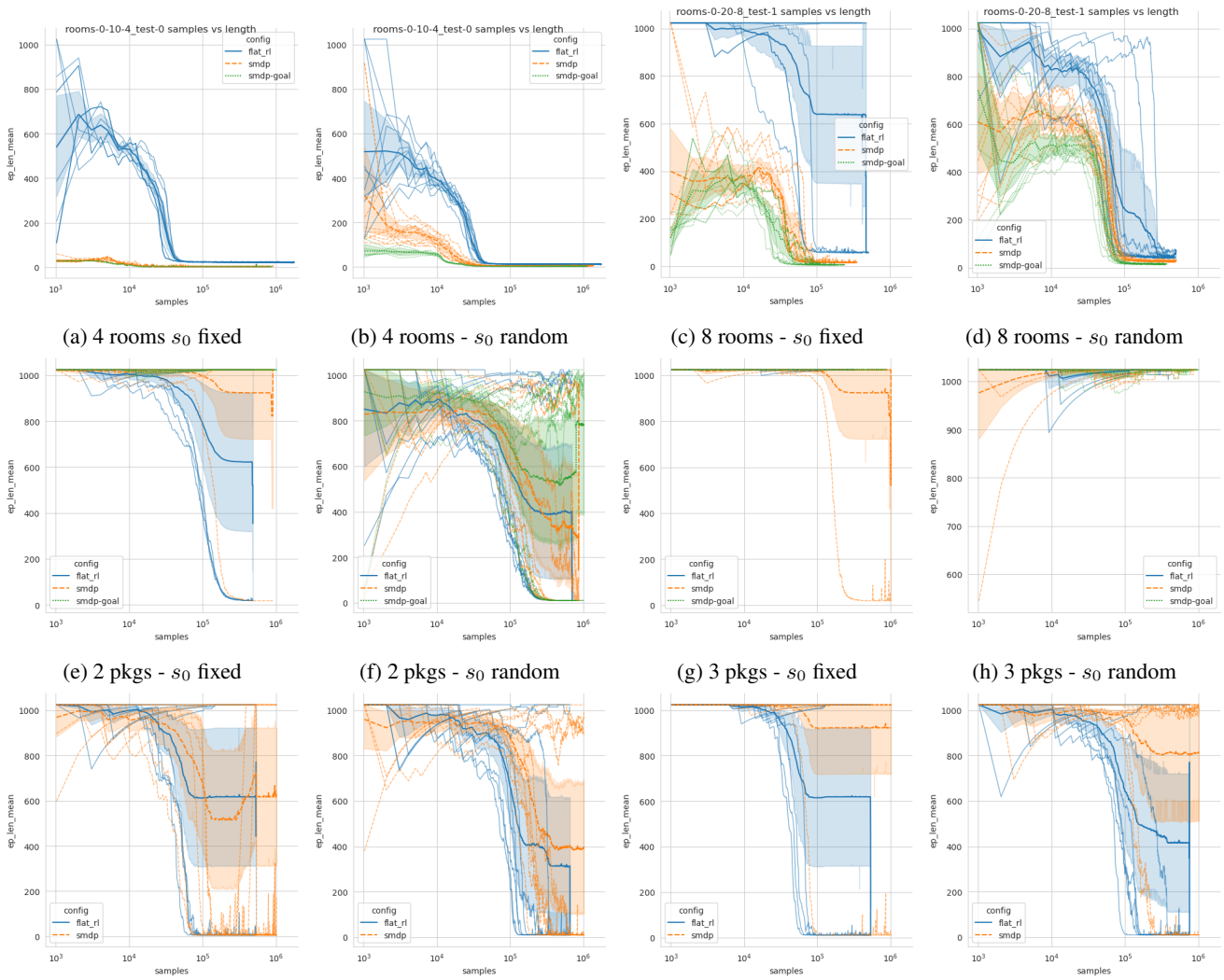
a mapping L_M from the feature vectors in $\mathcal{M}$ encoding the coordinate on the grid to the boolean vectors of the ground propositions in Π_M. (Both the domain file and the problem files can be found in Appendix A.1.) It is commonly understood that the options framework will improve the sample efficiency of the entire learning process if each individual option captures a meaningful subproblem such that the meta-learner using options can easily connect options to solve the complicated RL tasks. Such a subtask should capture the movement between rooms in the rooms domain.

Example 1 *In the 8 rooms domain on the 20×20 grid, the PPaRL task captures total 40 options that define all the movements from one room to its adjacent corridor. For a concrete example, we see an option O that's defining a movement from the room $r5$ to the corridor connecting to the room $r3$.*

$$I_O = \{s \in \mathcal{S} | \textit{in-room(r5:room)} \subseteq L(s)\}$$
$$\beta_O = \{s \in \mathcal{S} | \textit{in-room(c-r5-r3:room)} \subseteq L(s)\},$$

where we abused notation for a subset of $\mathcal{S}'$ that entails the logical expression such as in-room(r5:room) with the expression itself.

Next, we present the learning curves from 3 configurations on two problems instances, one from the 4 rooms on 10x10 girds and the other 8 rooms on 20x20 grids. In all training phases, we gave reward 100 when reaching the goal,



(a) 4 rooms s_0 fixed (b) 4 rooms - s_0 random (c) 8 rooms - s_0 fixed (d) 8 rooms - s_0 random

(e) 2 pkgs - s_0 fixed (f) 2 pkgs - s_0 random (g) 3 pkgs - s_0 fixed (h) 3 pkgs - s_0 random

(i) 4 blocks - s_0 fixed (j) 4 blocks - s_0 random (k) 4 blocks - s_0 fixed (l) 4 blocks - s_0 random

Figure 2: Learning curves showing the average length of the plans over the 10^6 samples from the same runs in Figure 1.

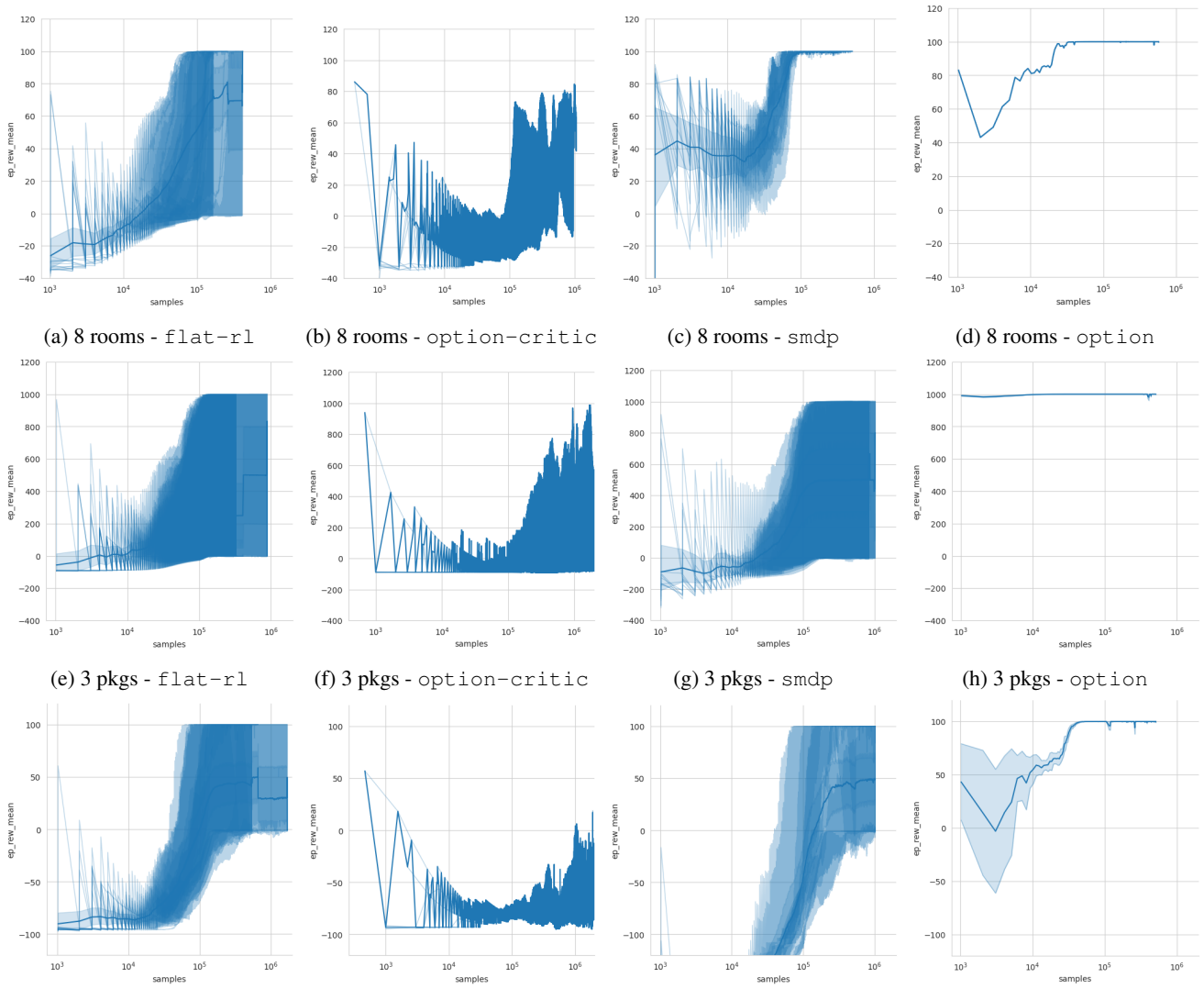
and we deduct a cost of -0.001 for each step, and cost -0.1 for applying inapplicable actions at each state. The discounting factor γ was set to 0.99, and the maximum episode length was limited to 1024. Since `smdp` uses pre-trained options derived from the PPaRL task, we provided a set of options that appear in the plan from the Π given the initial state and the goal in each evaluated task. Namely, we extended the action space with 3 options in the 4 rooms problem and 6 options in the 8 rooms problem. In addition to the options derived from Π, `smdp-goal` extends its set of options with the options derived from the goal state $\mathcal{I}_{O_o^{s_*}}$, which adds additional 2 more options for both problems.

In Figure (1a) and (1b), we can see that the learning curves from `smdp` (orange) and `smdp-goal` (green) reaches higher reward much earlier than `flat-rl` (blue). On the other hand, `flat-rl` achieves the highest reward in longer run, which is consistent to the SMDP learning theory (Sutton, Precup, and Singh 1999). We also see that `smdp` and `smdp-goal` converge much faster when we fixed the initial state (Figure (1a)) as defined in PDDL problem instance since we provided a set of options that covers the operators in a plan in Π from the fixed initial state. Figure (1c) and (1d) are obtained from 8 rooms problem instances and we observe the similar trend.

Figure (2) shows the decrease in the average length of the plan during training. From Figure (2a) to figure (2d), we can see that the `smdp` and `smdp-goal` configurations reached to the shorter length plans much earlier than `flat-rl` when they used an option as a macro action. Therefore, we can conclude that the options framework derived from the PPaRL task offers the options with the desired subtasks and improves the sample efficiency.

PDDL based domains

Our experiments include the two most famous PDDL based domains, `logistics` and `blocksworld` (4ops version).



(a) 8 rooms - `flat-rl` (b) 8 rooms - `option-critic` (c) 8 rooms - `smdp` (d) 8 rooms - `option`

(e) 3 pkgs - `flat-rl` (f) 3 pkgs - `option-critic` (g) 3 pkgs - `smdp` (h) 3 pkgs - `option`

(i) 4 blocks - `flat-rl` (j) 4 blocks - `option-critic` (k) 4 blocks - `smdp` (l) 4 blocks - `option`

Figure 3: Learning curves showing the average rewards over the 10^6 samples from `flat-rl`, `option-critic`, `smdp`, and the offline `option` training using `flat-rl` from the left to the right on the 8-`rooms`, `logistics`, and `blocksworld` domains.

The main reason that these two domains were chosen is the different result on the PPaRL task after applying the projection on the goal variables. In `logistics` domain, Π generates options that correspond to the temporally extended actions for individual packages. However, the `blocksworld` does not reduce the number of action operators from Π_M to Π. Since our framework is based on PDDLGym, our states are represented by boolean vectors of the size equivalent to the number of ground propositions. The shortcoming of such a representation is that the Deep RL algorithms do not scale well since the number of ground propositions tends to grow quickly. In addition, the number of ground actions also grows quickly compared with typical RL environments. Due to the limitations mentioned above, we experiment with small size problems. The PDDL instances can be found in Appendices A.2 and A.3.

Logistics Domains In `logistics` domain, we compare the configurations on domains having 2 cities with 2 packages and 3 cities with 3 packages. We first examine typical options derived from the PPaRL task.

Example 2 *In logistics domain, the objective is to move packages from the initial location to the goal location by driving a truck that connects a city to the airport in the city or by flying from one airport to the other using an airplane. We can see options that define loading or unloading actions for a package. The following option corresponds to an operator loading the package* $obj11$ *to the airplane* $apn1$ *at the*

airport $apt1$,

$$I_O = \{s \in \mathcal{S} | \textit{at(obj11:package,apt1:airport)} \subseteq L(s)\}$$
$$\beta_O = \{s \in \mathcal{S} | \textit{in(obj11:package,apn1:airplane)} \subseteq L(s)\}.$$

The learning curves from Figure (1e) to Figure (1h) present the average rewards from the evaluated configurations. During training, we gave reward 1000 when reaching the goal, and other parameters remained the same as the rooms domain. Figure (1g) and (1h) are obtained from an instance with 3 cities and 3 packages, and we see that both `flat-rl` and `smdp` do not reach the goal during training most of the trials (the reward remains negative). Yet, `smdp` reached to the goal with very long plans as shown in Figure (1g) when others completely failed. From Figure (1e) and (1f), we see that `flat-rl` and `smdp` were often successful to train policy functions reaching the goal, but we don't see clear advantage of using options. In logistics domain with 2 packages and 3 packages, the number of primitive actions in PDDLGym environment is 42 and 72, which may challenge PPO algorithm. Although individual options may define temporally extended actions for each package that might help decomposing the overall task, the high branching factor of the actions will likely to lead the rollout process to irrelevant state space.

Blocksworld Domains In `blocks` domain, we tested our algorithms on two problem instances with 4 blocks. Since the PPaRL task only abstracts away state variables for the grounded predicates from (clear ?x - block), the options do not extract useful subtasks. For example, the option O derived from (pick-up a) defines

$$I_O = \{s \in \mathcal{S} | \text{ontable(a:block)} \subseteq L(s)\},$$
$$\beta_O = \{s \in \mathcal{S} | \text{holding(a:block)} \subseteq L(s)\}.$$

This option supposed to encapsulate primitive actions that need to remove all the blocks on the block a and putting down any block at hand. Even if we were successful at learning those options, we expect that chaining those options won't help solving the original problem. Furthermore, the intra-option policy training for individual goal options $\mathcal{I}_{O_o^{s*}}$ was no easier than solving the original problem in contrast with `logistics` domain. The learning curves from Figure (1i) to Figure (1l) present the average rewards from `flat-rl` and `smdp`, where we provided the same rewards and hyper parameters used in the rooms domain. For the average length of the plan as shown in Figure (2i) to Figure (2l), we observe a similar trend in the `logistics` domain. Overall, we see that `smdp` converges slower than `flat-rl`.

Comparing All In Figure (3), we present the learning curves from the `flat-rl`, and `option-critic`, `smdp`, and the intra-policy learning. Each row from the top to the bottom shows the learning curves from the 8 rooms domain, logistics domain, and the blocksworld domain, respectively. We first see that the option training converges fastest as desired. Note that the `option-critic` is the worst compared with other two PPO based methods `flat-rl` and `smdp`, which is consistent to the reporting in (Zhang and Whiteson 2019).

Challenges

The current implementation based on PDDLGym poses several challenges, to be tackled in future work. First, the states of $\mathcal{M}$ are represented as boolean vectors, corresponding the truth assignments to the planning task ground propositions, obtained by a naive grounding of all lifted predicates, including static predicates. Thus, each state consists of a large portion of information that does not change from one state to another. Second, the action space in PDDLGym consists of all planning ground operators, regardless of their applicability in a state. Thus, randomly choosing an action often results in inapplicable actions. Third, the number of options obtained from the planning operators, even after filtering, is still quite large for the existing RL algorithms. Fourth, the `smdp` algorithm treats all options and actions equally, not giving preferences to options over atomic actions.

Related Work

Some of the relevant works that attempt to combine symbolic planning and RL include PEORL (Yang et al. 2018), SDRL (Lyu et al. 2019), and Taskable RL (Illanes et al. 2020). Our approach is similar to Taskable RL, but we do not require any explicit definition of options from the manually designed tasks. PEORL also considers integrating symbolic planning and hierarchical RL, however, it assumes that an exact representation of a MDP is available in the planning task and an option per transition is directly mapped from planning task transition system, which could be unrealistic. It updates value function and associated option policies only when options terminate, while our method can operate online to accommodate intra-option update. Moreover, PEORL is based on tabular representation, while our method is based on deep neural network representation. Although, SDRL is the deep learning extension of PEORL, it applies R-learning, whereas our method can use any RL algorithm.

Conclusions and Future Work

In this work, we have presented a simple general framework for annotating reinforcement learning tasks with planning tasks, to facilitate the transfer of planning based techniques into the field of reinforcement learning. We have shown how to define options based on planning operator description, and how to filter options based on a plan. Our preliminary experiments show the feasibility of our approach.

This, however, is not the end of the road. While our framework is generic, our initial implementation is based on PDDLGym. Our very next step is therefore to remove this dependency, allowing us to attempt to solve existing environments where a partial PDDL model exists, such as TextWorld (Côté et al. 2019) or Montezuma's Revenge (Qiu, Yao, and Wange 2018). Further, while this work focused on presenting our framework and definition of options, our exploitation of options was quite naive, using an existing RL algorithm. One can envision a plethora of planning based approaches that guide the RL agent. One such approach was suggested by Illanes et al. (2020), other approaches may include a re-planning based approach (Yoon, Fern, and Givan 2007).

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS action models with classical planning. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS^+ Planning. *Computational Intelligence* 11(4): 625–655.

Bacon, P.; Harb, J.; and Precup, D. 2017. The Option-Critic Architecture. In Singh, S.; and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 1726–1734. AAAI Press.

Barto, A. G.; and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems* 13(1): 41–77.

Côté, M.-A.; Kádár, Á.; Yuan, X.; Kybartas, B.; Barnes, T.; Fine, E.; Moore, J.; Hausknecht, M.; El Asri, L.; Adada, M.; Tay, W.; and Trischler, A. 2019. TextWorld: A Learning Environment for Text-Based Games. In Cazenave, T.; Saffidine, A.; and Sturtevant, N., eds., *Computer Games*, 41–75. Cham: Springer International Publishing. ISBN 978-3-030-24337-1.

Dietterich, T. G. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research* 13: 227–303.

Dulac-Arnold, G.; Mankowitz, D.; and Hester, T. 2019. Challenges of Real-World Reinforcement Learning.

Edelkamp, S. 2001. Planning with Pattern Databases. In Cesta, A.; and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

Eysenbach, B.; Salakhutdinov, R.; and Levine, S. 2019. Search on the Replay Buffer: Bridging Planning and Reinforcement Learning. In *Proceedings of the Thirtythird Annual Conference on Neural Information Processing Systems (NIPS 2019)*, 15220–15231.

Grounds, M. J.; and Kudenko, D. 2007. Combining Reinforcement Learning with Symbolic Planning. In Durfee, E.; and Yokoo, M., eds., *Proceedings of the 6th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2007)*, volume 4865, 75–86. IFAAMAS/ACM.

Harb, J.; Bacon, P.; Klissarov, M.; and Precup, D. 2018. When Waiting Is Not an Option: Learning Options With a Deliberation Cost. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 3165–3172. AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 176–183. AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of the ACM* 61(3): 16:1–63.

Hoffmann, J.; Sabharwal, A.; and Domshlak, C. 2006. Friends or Foes? An AI Planning Perspective on Abstraction and Search. In Long, D.; Smith, S. F.; Borrajo, D.; and McCluskey, L., eds., *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, 294–303. AAAI Press.

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic Plans as High-Level Instructions for Reinforcement Learning. In Beck, J. C.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, 540–550. AAAI Press.

Katz, M.; and Domshlak, C. 2010. Implicit Abstraction Heuristics. *Journal of Artificial Intelligence Research* 39: 51–126.

Kazemitabar, S. J.; and Beigy, H. 2008. Automatic Discovery of Subgoals in Reinforcement Learning using Strongly Connected Components. In *Proceedings of the Twenty-second Annual Conference on Neural Information Processing Systems (NIPS 2008)*, 829–834.

Konda, V. R.; and Tsitsiklis, J. N. 2000. Actor-Critic Algorithms. In *Proceedings of the Fourteenth Annual Conference on Neural Information Processing Systems (NIPS 2000)*, 1008–1014.

Konidaris, G.; and Barto, A. 2009. Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining. In *Proceedings of the Twenty-third Annual Conference on Neural Information Processing Systems (NIPS 2009)*, 2295–2304.

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

Lyu, D.; Yang, F.; Liu, B.; and Gustafson, S. 2019. SDRL: Interpretable and Data-efficient Deep Reinforcement Learning Leveraging Symbolic Planning. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 2970–2977. AAAI Press.

Machado, M. C.; Bellemare, M. G.; and Bowling, M. 2017. A Laplacian Framework for Option Discovery in Reinforcement Learning. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML 2017)*, 2295–2304.

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

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari With Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*.

Qiu, D.; Yao, S.; and Wange, W. 2018. Deep Reinforcement Learning with Abstract High-level Symbolic

Planning (Technical Report). Technical report. Available at https://www.researchgate.net/profile/Dicong_Qiu2/publication/338174994_Deep_Reinforcement_Learning_with_Abstract_High-level_Symbolic_Planning/links/5e04c18f4585159aa49b14ad/Deep-Reinforcement-Learning-with-Abstract-High-level-Symbolic-Planning.pdf.

Raffin, A.; Hill, A.; Ernestus, M.; Gleave, A.; Kanervisto, A.; and Dormann, N. 2019. Stable baselines3. URL https://github.com/DLR-RM/stable-baselines3.

Ramesh, R.; Tomar, M.; and Ravindran, B. 2019. Successor Options: An option Discovery Algorithm for Reinforcement Learning .

Riemer, M.; Liu, M.; and Tesauro, G. 2018. Learning Abstract Options. In *Proceedings of the Thirtysecond Annual Conference on Neural Information Processing Systems (NIPS 2018)*, 10445–10455.

Ryan, M. R. K. 2002. Using Abstract Models of Behaviours to Automatically Generate Reinforcement Learning Hierarchies. In Sammut, C.; and Hoffmann, A. G., eds., *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, 522–529. ACM.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347. URL http://arxiv.org/abs/1707.06347.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized Label Reduction for Merge-and-Shrink Heuristics. In Brodley, C. E.; and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2358–2366. AAAI Press.

Silver, T.; and Chitnis, R. 2020. PDDLGym: Gym Environments from PDDL Problems. In *ICAPS 2020 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*.

Stolle, M.; and Precup, D. 2002. Learning Options in Reinforcement Learning. In *International Symposium on abstraction, reformulation, and approximation*, 212–223.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence* 112(1-2): 181–211.

Torralba, Á.; and Sievers, S. 2019. Merge-and-Shrink Task Reformulation for Classical Planning. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5644–5652. IJCAI.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning* 8(3–4): 279–292.

Yang, F.; Lyu, D.; Liu, B.; and Gustafson, S. 2018. PEORL: Integrating Symbolic Planning and Hierarchical Reinforcment Learning for Robust Decision-Making. In Lang, J., ed., *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, 4860–4866. IJCAI.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 352–360. AAAI Press.

Zhang, S. 2018. Modularized Implementation of Deep RL Algorithms in PyTorch. URL https://github.com/ShangtongZhang/DeepRL.

Zhang, S.; and Whiteson, S. 2019. DAC: The Double Actor-Critic Architecture for Learning Options. In *Proceedings of the Thirty-third Annual Conference on Neural Information Processing Systems (NeurIPS 2019)*.

A Appendix

A.1 N-rooms PDDL

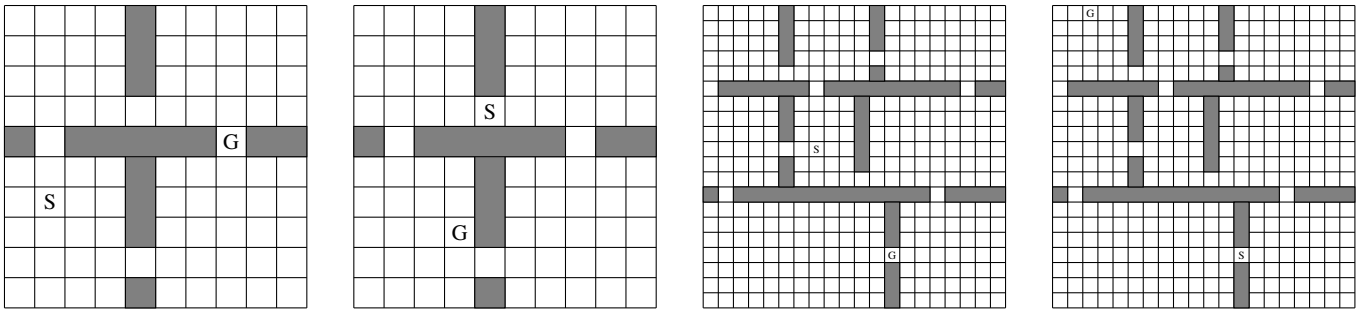
The PDDL domain file used for the n-rooms problem is described in what follows.

```
(define (domain rooms)
  (:requirements :strips :typing :negative-preconditions)
    (:types
        location - object
        room - object
    )
    (:predicates
        (at ?x - location)
        (in-room ?r - room)
        (IN ?x - location ?r - room)
        (CONNECTED ?x - location ?y - location)
        (CONNECTED-ROOMS ?r - room ?s - room)
    )

    (:action move-in-room
        :parameters (?from - location ?to - location ?r - room)
        :precondition (and
            (IN ?from ?r)
            (IN ?to ?r)
            (CONNECTED ?from ?to)
            (in-room ?r)
            (at ?from)
        )
        :effect (and
            (not (at ?from))
            (at ?to)
        )
    )

    (:action move-out-room
        :parameters (?from - location ?to - location ?r - room ?s - room)
        :precondition (and
            (IN ?from ?r)
            (IN ?to ?s)
            (CONNECTED ?from ?to)
            (CONNECTED-ROOMS ?r ?s)
            (at ?from)
            (in-room ?r)
            (not (at ?to))
            (not (in-room ?s))
        )
        :effect (and
            (not (at ?from))
            (at ?to)
            (not (in-room ?r) )
            (in-room ?s)
        )
    )
)
```

Next, we visualize the problem files. These problem files have a large number of static predicates that describe the room layout, and are rather straightforward to derive. The layout is shown in Figure 4.



(a) (b) (c) (d)

Figure 4: Problem files layout for the four instances of the n-rooms domain.

A.2 Logistics PDDL instances

```
(define (problem logistics-c2-p2-0)
(:domain logistics)
(:objects
apn1 - airplane
apt1 apt2 - airport
pos2 pos1 - location
cit2 cit1 - city
tru2 tru1 - truck
obj21 obj11 - package)

(:init
    (at apn1 apt2)
    (at tru1 pos1)
    (at obj11 pos1)
    (at tru2 pos2)
    (at obj21 pos2)
    (IN-CITY pos1 cit1)
    (IN-CITY apt1 cit1)
    (IN-CITY pos2 cit2)
    (IN-CITY apt2 cit2))

(:goal (and
            (at obj11 pos2)
            (at obj21 pos1)
        ))
)
```

```
(define (problem logistics-c2-p2-1)
(:domain logistics)
(:objects
 apn1 - airplane
 apt1 apt2 - airport
 pos2 pos1 - location
 cit2 cit1 - city
 tru2 tru1 - truck
 obj21 obj11 - package)

(:init
    (at apn1 apt2)
    (at tru1 pos1)
    (at obj11 pos1)
    (at tru2 pos2)
```

```
    (at obj21 pos2)
    (IN-CITY pos1 cit1)
    (IN-CITY apt1 cit1)
    (IN-CITY pos2 cit2)
    (IN-CITY apt2 cit2))

(:goal (and
            (at obj11 apt1)
            (at obj21 pos1)
        ))
)
(define (problem logistics-c3-p3_test-0)
(:domain logistics)
(:objects
  apn1 - airplane
  apt3 apt2 apt1 - airport
  pos3 pos2 pos1 - location
  cit3 cit2 cit1 - city
  tru3 tru2 tru1 - truck
  obj31 obj21 obj11 - package)

(:init
    (at apn1 apt1)
    (at tru1 pos1)
    (at obj11 pos1)
    (at tru2 pos2)
    (at obj21 pos2)
    (at tru3 pos3)
    (at obj31 pos3)
    (IN-CITY pos1 cit1)
    (IN-CITY apt1 cit1)
    (IN-CITY pos2 cit2)
    (IN-CITY apt2 cit2)
    (IN-CITY pos3 cit3)
    (IN-CITY apt3 cit3))

(:goal (and
            (at obj11 apt1)
            (at obj21 apt2)
            (at obj31 apt3)
        ))
)


(define (problem logistics-c3-p3_test-1)
(:domain logistics)
(:objects
  apn1 - airplane
  apt3 apt2 apt1 - airport
  pos3 pos2 pos1 - location
  cit3 cit2 cit1 - city
  tru3 tru2 tru1 - truck
  obj31 obj21 obj11 - package)

(:init
    (at apn1 apt1)
    (at tru1 pos1)
    (at obj11 pos1)
    (at tru2 pos2)
```

```
    (at obj21 pos2)
    (at tru3 pos3)
    (at obj31 pos3)
    (IN-CITY pos1 cit1)
    (IN-CITY apt1 cit1)
    (IN-CITY pos2 cit2)
    (IN-CITY apt2 cit2)
    (IN-CITY pos3 cit3)
    (IN-CITY apt3 cit3))

(:goal (and
            (at obj11 apt2)
            (at obj21 pos1)
            (at obj31 apt3)
        ))
)
```

A.3 BlocksWorld PDDL instances

```
(define (problem BLOCKS-4-0)
(:domain BLOCKS)
(:objects D B A C - block)
(:INIT (CLEAR C) (CLEAR A) (CLEAR B) (CLEAR D) (ONTABLE C) (ONTABLE A)
 (ONTABLE B) (ONTABLE D) (HANDEMPTY))
(:goal (AND (ON D C) (ON C B) (ON B A) (ONTABLE A)))
)
```

```
(define (problem BLOCKS-4-1)
(:domain BLOCKS)
(:objects A C D B - block)
(:INIT (CLEAR B) (ONTABLE D) (ON B C) (ON C A) (ON A D) (HANDEMPTY))
(:goal (AND (ON D C) (ON C A) (ON A B)))
)
```