Leveraging Symbolic Planning Models in Hierarchical Reinforcement Learning

León Illanes[1,2], **Xi Yan**[1], **Rodrigo Toro Icarte**[1,2], **Sheila A. McIlraith**[1,2]
[1]University of Toronto, [2]Vector Institute
Toronto, Ontario, Canada
`{lillanes, rntoro, sheila}@cs.toronto.edu, xi.yan@mail.utoronto.ca`

Abstract

We investigate the use of explicit action models—as typically used for Automated Planning—in the context of Reinforcement Learning (RL). These action models allow agents to reason about macro-actions and high-level symbolic state spaces. As a consequence, agents with access to an action model and a planner can automatically synthesize high-level plans that can, in turn, be used as high-level instructions to significantly improve sample efficiency. Our approach is based on classical and partial-order planning, in combination with hierarchical RL and recent advances in reward specification and problem decomposition for RL. Empirical results show that our approach finds high-quality policies for previously unseen tasks in extremely few training steps, consistently outperforming standard Hierarchical RL techniques.

1 Introduction

Reinforcement learning (RL) techniques allow for agents to perform tasks in complex domains, where environment dynamics and reward structures are initially unknown. These techniques are based on performing random exploration, observing the dynamics and reward returned by the environment, and synthesizing an optimal policy that maximizes expected cumulative reward. Unfortunately, when reward is sparsely distributed, as is the case in many applications, RL techniques can suffer from poor *sample efficiency*, requiring millions of episodes to learn reasonable policies. Further, these systems are typically not *taskable*: specifying new tasks is often difficult and the skills that are learned for one task are not easily transferred to others. Over the years a number of approaches have been proposed to address these shortcomings including efforts to learn hierarchical representations [5], to define *options* or macro-actions [23] that can be used by the RL system, or to learn skills that are somehow independent of the state space where they were learned [12].

Our interest in this paper is in leveraging high-level symbolic planning models and automated plan synthesis techniques, in concert with state-of-the-art RL techniques, with the objective of significantly improving sample efficiency and creating systems that are human taskable. Our efforts are based on the observation that some approximated understanding of the environment can be characterized as a symbolic planning model—a set of properties of the world and a formal description of actions that cause those properties to change in predictable ways.

Recent research has demonstrated the benefit of providing high-level instructions to an RL system to improve sample efficiency (e.g., policy sketches [1], LTL advice [25], reward machines [26]). Nevertheless, these instructions must be *manually* generated. By using a symbolic model of the environment rather than a task-specific set of instructions we are able to *automatically* generate instructions in the form of a sequential or partial-order plan—the latter compactly characterizing a multitude of sequential plans. Such a plan is then used to enhance a Hierarchical Reinforcement Learning (HRL) system by ignoring options that do not represent advancement in the plan.

KR2ML Workshop at NeurIPS 2019, Vancouver, Canada.

We compare our approach to standard forms of HRL. Our results show that the approach is an effective method for specifying tasks to an RL agent, reaching high-quality policies for previously unseen tasks in extremely few training steps.

2 Related Work

The general idea of modeling a problem at various levels of abstraction and then taking advantage of different reasoning capabilities at each level forms the basis of HRL [21, 4, 24, 18, 23, 5]. The specific use of symbolic planning techniques as a method for reasoning at the high-level while using Reinforcement Learning techniques at the lower level has been an active research topic for decades [18, 8, 9, 29, 28, 14]. In this section, we briefly describe some of these works and highlight some of the key differences with our proposed approach.

The options framework [23] has become a well-known standard approach for exploiting temporal abstraction in Reinforcement Learning. Our work is based on this framework, and specific details regarding how we formalize the framework for our purposes will be given in Section 3. A key contribution of our work is the way in which we use explicit symbolic planning to do online filtering of the set of available options.

Other existing approaches have used symbolic planning to select options or macro-actions. An early approach proposed using a symbolic planner coupled into an RL agent [8]. There, the planner produces an initial high-level plan and is subsequently used to replan when the plan's preconditions are violated. A related approach uses a sequential plan to modify the reward via reward shaping [9]. In contrast to these approaches, our system uses planners to produce a single and final plan that is then used to define a reward function. The structure in the plan can be exploited to produce an easily decomposable reward function.

Recent work has also proposed methods based on coupling a planner to an RL agent [28, 14]. There, the focus has been on two-way communication between agent and planner to continuously improve the high-level model and find better high-level solutions. In our case, we assume an adequate high-level model is given and we show how to exploit it. Other work has explored techniques for automatically building such abstractions [13, 11]. Integrating such techniques into our work may prove to be an interesting direction for future work.

Finally, there has also been work that has focused on learning explicit state-transition systems that represent high-level models [29]. With these, standard graph search algorithms can be used to find sequences of macro-actions. Our work considers *implicit* state-transition systems described as classical planning domains. This allows us to consider highly combinatorial problems that correspond to state-transition systems that are far too large to represent explicitly.

3 Preliminaries

In this section we establish relevant notation and review key aspects of reinforcement learning and automated planning. In addition, we describe a simple running example.

3.1 Reinforcement Learning

For the purposes of this work, we will say that the environment in which an RL agent acts is formalized as a tuple $\mathcal{E} = \langle S, A, p \rangle$, where S is its set of states, A is the set of available actions, and $p(s_{t+1} \mid s_t, a_t)$ is the transition probability distribution. A policy is defined as a probability distribution $\pi(a \mid s)$ that establishes the probability of the agent taking action a given that its current state is s.

A task T for environment $\mathcal{E}$ is defined as an MDP with the states and transitions of $\mathcal{E}$, reward function $r \colon S \times A \times S \to \mathbb{R}$, and discount factor $\gamma \in (0, 1]$. Then, the objective is to find a policy that maximizes the expected discounted reward from every state $s \in S$.

3.1.1 Reward Machines

A reward machine is a finite-state machine that can be used to specify temporally-extended and non-Markovian reward functions. The intuitive idea is that transitions in the reward machine take place based on observations made by the agent about the environment, and that the reward depends on the transitions taken in the machine. The observations are represented by a set of propositional symbols $\mathcal{P}$, which correspond to facts that the agent may perceive. For a given environment, we assume there is a labeling function $L\colon S \to 2^{\mathcal{P}}$ that establishes what is perceived when reaching a state. Then, a reward machine for environment $\mathcal{E}$ and observation propositions $\mathcal{P}$ is given by the tuple $\mathcal{R} = \langle U, u_0, \delta_u, \delta_r \rangle$. U is its set of states, u_0 is its initial state, $\delta_u\colon U \times 2^{\mathcal{P}} \to U$ is its state transition function, and $\delta_r\colon U \times U \to \mathbb{R}$ is its reward transition function.[1] Whenever the agent makes a transition (s, a, s') in the environment and observes $P \subseteq \mathcal{P}$, the current state in the reward machine is updated from u to $u' = \delta_u(u, P)$. At this point, the agent receives reward $\delta_r(u, u')$.

3.1.2 Temporal Abstractions for Reinforcement Learning

Standard RL techniques are faced with significant issues when applied on large-scale problems. In practical terms, RL algorithms need a large amount of training steps in order to converge. A popular technique for dealing with these issues is to consider temporally-extended macro-actions that represent useful high-level behaviors. In particular, the options framework proposes the use of policies that are trained for achieving specific high-level behaviors, coupled with well-defined criteria for their termination [23]. Given the current state, an agent acting within this framework chooses one among the high-level options and executes its policy until it terminates.

For a given environment $\mathcal{E} = \langle S, A, p \rangle$ and labeling function $L\colon S \to 2^{\mathcal{P}}$, we formalize the notion of an option as $o = \langle \pi_o, T_o \rangle$ where π_o is the option's policy and $T_o \subseteq \mathcal{P}$ defines its termination condition. The application of o consists of following policy π_o until reaching some state $s' \in S$ such that $T_o \subseteq L(s')$. After this, some other option can be selected, or the execution can terminate.

3.2 Symbolic Planning

We specify planning domains in terms of a tuple $\mathcal{D} = \langle \mathcal{F}, \mathcal{A} \rangle$. $\mathcal{F}$ is a set of propositional symbols, called the fluents of $\mathcal{D}$, and $\mathcal{A}$ is the set of planning actions in the domain. Planning states are specified as subsets of $\mathcal{F}$, so that state $\mathcal{S} \subseteq \mathcal{F}$ represents the situation in which the fluents in $\mathcal{S}$ are all true and those not in $\mathcal{S}$ are false. An action $a \in \mathcal{A}$ is specified in terms of its preconditions and effects, which are given as logical formulae over the propositional fluents. Actions are only applicable in states where their preconditions are satisfied, and such application results in a transition to a state where the action's effects are true, and everything else remains unchanged.

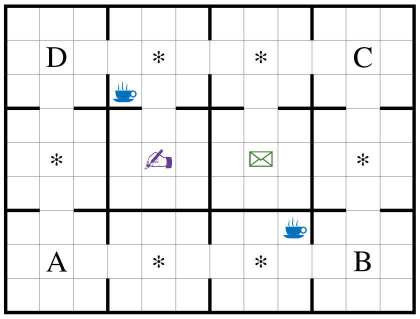
A planning task is described by an initial state and a goal condition. The goal is given as a formula over the fluents of the domain. Any state that satisfies the goal condition is said to be a goal state. A sequence of actions $\Pi = [a_0, a_1, \dots, a_n]$ is known as a sequential plan for a task when it is possible to sequentially apply the actions starting at the initial state, and doing so reaches a goal state. Given a plan $\Pi = [a_0, a_1, \dots, a_n]$, we will refer to its prefix with respect to action a_i as $\textbf{prefix}(\Pi, a_i) = [a_0, a_1, \dots, a_{i-1}]$.

Partial-order plans generalize sequential plans by relaxing the ordering condition over the actions. A partial-order plan is a tuple $\overline{\Pi} = \langle \overline{\mathcal{A}}, \prec \rangle$, where $\overline{\mathcal{A}}$ is its set of action occurrences and $\prec$ is a partial order over $\overline{\mathcal{A}}$. The set of linearizations of $\overline{\Pi}$, denoted $\Lambda(\overline{\Pi})$, is the set of all sequences of the action occurrences in $\overline{\mathcal{A}}$ that respect the partial order $\prec$. Any linearization $\Pi \in \Lambda(\overline{\Pi})$ is a sequential plan for the task. Intuitively, a partial-order plan represents a family of related sequential plans. Note that it is entirely possible for a plan to require using the same action twice. As such, two action occurrences in $\overline{\mathcal{A}}$ may be repetitions of the same action, distinguished only for the purposes of defining the particular partial-order plan.

3.3 Running Example

We consider a version of the OFFICEWORLD domain described by Toro Icarte et al. (2018). The low-level environment is represented by the grid displayed in Figure 1a. An agent situated in any cell can try to move in any of the four cardinal directions, succeeding only if the movement does not go through a wall. The symbols in the grid represent events that can be perceived by the agent: it picks up coffee or mail when it reaches the locations marked with blue cups or the green envelope,

[1]Since we limit ourselves to using simple reward machines, this definition is slightly different (but equivalent) to the one given by Toro Icarte et al. (2018), which is based on the definition of full-fledged reward machines.



(a) The OFFICEWORLD grid environment. Different symbols represent the events perceived when reaching particular locations.

`get-coffee`:
preconditions: (none)
effects: `have-coffee`
observations: *`coffee-machine`*
`deliver-coffee`:
preconditions: `have-coffee`
effects: `delivered-coffee`, **not** `have-coffee`
observations: *`office`*

(b) Example planning actions in the OFFICEWORLD.

Figure 1: The OFFICEWORLD running example.

respectively, or it can deliver what it picked up by reaching the office, marked by the purple writing hand, etc. The locations marked ✻ are places the agent must not step on. The four additional named locations (A, B, C, D) can also be recognized by the agent. An example of a task is that of delivering both mail and coffee to the office. For this task, any optimal policy will need to choose whether to get the coffee or mail first depending on the agent's initial position.

Figure 1b shows two high-level actions and their preconditions and effects. In addition, it shows the low-level observations that should be perceived when the actions are executed. Note, for example, that the only observation associated with the `deliver-coffee` action is *`office`*, and that this makes no reference to the coffee itself. Whether or not the agent is holding coffee is a high-level fact that cannot be directly observed by looking only at the current low-level state. Instead, it is a state property that depends on the past trajectory of the robot: the robot is holding coffee only if it has visited the coffee machine and has not visited the office since.

4 Planning Models in RL

Given a low-level environment $\mathcal{E} = \langle S, A, p\rangle$ and the set of symbols $\mathcal{P}$ that represent possible observations, a symbolic model for $\mathcal{E}$ is specified as $\mathcal{M} = \langle \mathcal{D}, \alpha\rangle$, where $\mathcal{D} = \langle \mathcal{F}, \mathcal{A}\rangle$ is a planning domain and $\alpha\colon \mathcal{A} \to 2^{\mathcal{P}}$ is a function that associates planning actions with sets of observations. The symbols in $\mathcal{F}$ do not directly map to the observations and they can—as in the above example with the robot and the coffee—be used to model state properties that cannot be perceived in the low-level environment. Some of the other symbols used in the OFFICEWORLD running example are meant to represent whether the robot is carrying mail, the set of locations it visited in the past, and whether it has already delivered coffee or mail to office.

Note that one of the key properties of automated planning systems based on symbolic models is that specifying individual tasks is very simple. We take advantage of this by enabling the description of such tasks—goals in the symbolic models—to be communicated to an RL agent.

4.1 From Symbolic Actions to Options

Given an environment $\mathcal{E}$ and a corresponding symbolic model $\mathcal{M}$, we can define a set of options that represents relevant transitions that can occur in $\mathcal{M}$. For every planning action $a \in \mathcal{A}$, we build a termination condition T_a that is satisfied by all the low-level states in which every observation associated with a is perceived. Formally, this is defined in terms of $L^{-1}\colon 2^{\mathcal{P}} \to 2^{S}$, the generalized or multivalued inverse of the labeling function: $T_a = L^{-1}(\alpha(a)) \subseteq S$. Then, the set of options can be defined as $O(\mathcal{M}) = \{\langle \pi_a, T_a\rangle \mid a \in \mathcal{A}\}$, where every π_a is a policy trained specifically for reaching the states in T_a. Note that two or more actions in $\mathcal{A}$ may produce the same target set and be represented by a single option. In the OFFICEWORLD example, the high-level actions for delivering mail, delivering coffee, and visiting the office all map to a single observation (*`office`*) and therefore to a single option whose policy is meant to move the robot to the office.

4.2 From Plans to Reward Machines

We propose that a good way of communicating the high-level tasks to the agent is not in terms of the goals of the task, but rather in terms of high-level plans that achieve them. This allows us to specify tasks that are non-Markovian from the perspective of the low-level environment, such as delivering coffee in the OFFICEWORLD, while simultaneously enabling a natural form of task decomposition. To give a specific high-level plan to an RL agent, we will design a simple reward machine that directly represents the execution of the plan. Reward of 1 will be given only upon completing this execution.

For a sequential plan $\Pi = [a_0, a_1, \dots, a_n]$, the implementation of the corresponding reward machine is very straightforward and shown in Figure 2a. If instead of a sequential plan we consider a partial-order plan $\overline{\Pi} = \langle \overline{\mathcal{A}}, \prec \rangle$, we need a reward machine that effectively represents all possible orderings of the plan, in a way that any execution of one of them results in receiving a reward of 1 only on the last step. We build this machine by including one state for every possible subset of $\overline{\mathcal{A}}$. This ensures that the machine's executions correctly keep track of which actions have already taken place. Note that some of these states will not be reachable. In practice, when constructing the reward machine we only generate the states that are needed (i.e., those that are reachable). The possible transitions will be defined by explicitly considering $\Lambda(\overline{\Pi})$, as described in Figure 2b.

- $U = \{u_0, u_1, \dots, u_{n+1}\}$
- $\delta_r(u_i, u_j) =$ $\begin{cases} 1 & \text{if } i = n \text{ and } j = n+1 \\ 0 & \text{in any other case} \end{cases}$
- $\delta_u(u_i, P) =$ $\begin{cases} u_{i+1} & \text{if } P = \alpha(a_i) \\ u_i & \text{in any other case} \end{cases}$

(a) Reward machine based on a sequential plan $\Pi = [a_0, a_1, \dots, a_n]$.

- $U = 2^{\overline{\mathcal{A}}}$, $u_0 = \varnothing$
- $\delta_r(u_i, u_j) = \begin{cases} 1 & \text{if } u_j = \overline{\mathcal{A}} \\ 0 & \text{in any other case} \end{cases}$
- $\delta_u(u, P) =$ $\begin{cases} u \cup \{a\} & \text{if } \exists a \in \overline{\mathcal{A}}, \Pi \in \Lambda(\overline{\Pi}), \\ & \quad \text{s.t. } u = \textbf{prefix}(\Pi, a) \\ & \quad \text{and } P = \alpha(a) \\ u & \text{in any other case} \end{cases}$

(b) Reward machine based on a partial-order plan $\overline{\Pi} = \langle \overline{\mathcal{A}}, \prec \rangle$.

Figure 2: Reward machines defined by plans.

Note that the given definition of δ_u does not necessarily imply a function. Consider two actions $a, b \in \overline{\mathcal{A}}$ such that $P = \alpha(a) = \alpha(b)$, that is, actions that result in the same observations. In the OFFICEWORLD, an action to simply visit the office achieves the same event as the action to deliver coffee. Now, if there are two linearizations of $\overline{\Pi}$, Π_a and Π_b, such that $u = \textbf{prefix}(\Pi_a, a) = \textbf{prefix}(\Pi_b, b)$, then either action could be used in the definition of $\delta_u(u, P)$. In such cases, we arbitrarily choose which of the actions to use, as either choice represents a valid linearization of $\overline{\Pi}$.

4.3 From Reward Machines to Meta-Controllers

Given an environment $\mathcal{E} = \langle S, A, p \rangle$ and a symbolic model $\mathcal{M} = \langle \mathcal{D}, \alpha \rangle$, we want to leverage the set of options $O(\mathcal{M})$ to efficiently find a low-level policy for achieving some high-level goal g. Following the process outlined in Section 4.2, we can obtain reward machines that represent plans—sequential or partial order—to achieve g. Subsequently, we can use any algorithm that can successfully exploit the structure given by the reward machine. Ostensibly, the natural approach would be to use QRM, the algorithm specifically designed by Toro Icarte et al. (2018) in order to exploit this structure. QRM works by learning distinct policies for each state in the machine. However, these policies are goal specific, and cannot be easily transferred to new tasks. To address this issue, we consider a simpler alternative that was introduced by Toro Icarte et al. as an evaluation baseline. The approach follows the standard HRL setting, which consists of using some RL algorithm to learn a meta-controller that selects options. The only addition is that the meta-controller is—at every step—restricted to select only among options that are reasonable given the structure and current state of the reward machine.

To formalize this notion, we consider a reward machine $\mathcal{R} = \langle U, u_0, \delta_u, \delta_r \rangle$ that represents the task at hand. We will define a function $\Omega \colon U \to 2^{O(\mathcal{M})}$ and will use it to restrict the options available to the agent. Specifically, the agent will only be able to use an option $o \in O(\mathcal{M})$ if the current reward machine state u is such that $o \in \Omega(u)$. The intention is to limit the agent to select only among high-level actions that can progress the reward machine towards a different state. Then, the definition of Ω follows from inspection of the reward machine's structure:

$$\Omega(u) = \{o \in O(\mathcal{M}) \mid \delta_u(u, P_o) \neq u\}$$

5 Empirical Evaluation

We evaluated our approach by considering three different low-level environments and respective high-level symbolic models. In each case, we defined options using the approach outlined in Section 4.1. To best evaluate the effectiveness at leveraging previous experience, we trained the options on small tasks and then measured the performance of our algorithms on new tasks for each environment. For each task we computed sequential and partial-order plans, and built the corresponding reward machines. By restricting the meta-controller to strictly follow the sequential plan we obtain a first approach, which we have called `naive` in the following discussion. If we use a partial-order plan instead, we can restrict the meta-controller to only choose high-level actions that would advance the state in the corresponding reward machine. This is our main approach, and is called $\texttt{HRL}_\texttt{r}$.

We compare the results obtained by our algorithms to those obtained by simple baselines that use the same reward functions and pretrained options, but in a standard HRL framework (i.e., without restricting the options). These are `HRL(seq)` and `HRL(pop)`, and respectively use the reward machines constructed out of sequential and partial-order plans, but only as black-box reward functions.

In tabular cases, all training was done through the Q-learning algorithm [27]. For the continuous domain, we used DQN [16].

Benchmark Environments The first test environment is the OFFICEWORLD running example. Here the high-level actions include visiting any of the named locations, getting coffee, getting mail, delivering either coffee or mail to the office, and delivering both coffee *and* mail to the office. This results in options for going to any of the named locations or to the office, and for getting coffee and getting mail. The actions for delivering to the office are all mapped to the same observation—the agent reaching the office—and therefore correspond to the same option. For this environment, we tested on tasks consisting of 7 different goals and 10 random initial states for each goal. Each individual experiment was run 10 times.

Our second environment is the Minecraft-inspired gridworld described by Andreas et al. [1]. The grid contains raw materials (e.g., wood, iron) and locations where the agent can combine materials to produce refined materials (e.g., wooden planks), tools (e.g., hammer), and goods (e.g., goldware). The high-level actions allow for collecting each of the raw materials, and for achieving the combinations. The types of tasks that we evaluated on include examples such as *"build a bridge"* or *"have a hammer and a pickaxe"*. We ran experiments on 10 different maps, 10 different final-state goals, and 3 random initial states for each combination of map and goal. Each such experiment was run 3 times.

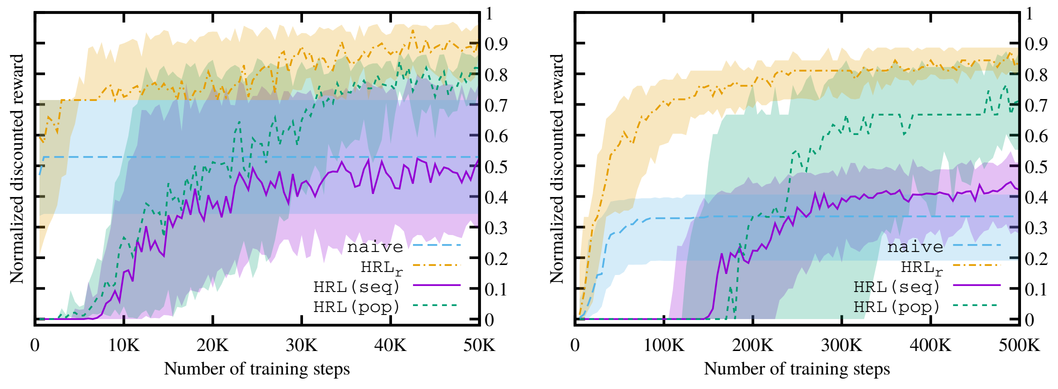
Our third environment, FARMWORLD, situates the agent in a continuous 2D map populated by a group of moving animals. The agent can control its own acceleration, and must collect resources from the animals. As in the previous domain, the high-level actions and tasks involve collecting specific resources (in this case by reaching certain animals) and combining them. Some processes involve consuming the animals, whereas others do not (e.g., a cow can be used to produce beef once, or to produce milk multiple times). We tested on 2 maps, 10 different goals, and 2 independent trials for each combination.

Pretraining Options In all environments, we pretrained the options by generating a small set of random initial states. We simultaneously trained a single policy for each option. The experience observed when training for any given option was used for training all other options, in an off-policy learning setting. This pretraining was restricted to a small number of reinforcement learning steps. In each independent experiment, option policies were continuously refined as the agent continued to interact with the environment.

Computing Plans For the computation of sequential plans we represented the problems in the Planning Domain Definition Language (PDDL) [15] and used the FastDownward planning system [10][2], in its LAMA configuration [19]. This allowed us to quickly produce high-quality sequential plans. In order to get partial-order plans, we relaxed the ordering in the sequential plans found by Fast Downward. Specifically, we used a mixed integer linear programming formulation proposed by Do and Kambhampati (2003) in the context of metric temporal planning, and subsequently adapted for classical planning problems by Muise et al. (2016)[3].

5.1 Results and Discussion

We report the results for the OfficeWorld and FarmWorld environments in Figure 3. Each graph displays the performance obtained after training with the labeled algorithm for the specified number of steps. We evaluated the results every 500 training episodes. In both cases, we report the normalized median discounted reward obtained on all experiments (using discount $\gamma = 0.9$) . We also show the median quintile performance (shaded area). Rewards were normalized by the best value obtained for each task across all independent experiments.



(a) OfficeWorld (b) MineCraftWorld

Figure 3: Experimental performance obtained on previously unseen tasks in two discrete domains. Median (and median quintile, as the shaded area) discounted reward normalized by best result obtained for each task across all experiments, using four different algorithms. The approaches that restrict option application to options that advance the plan (`naive` and $\texttt{HRL}_\texttt{r}$) converge to high-quality policies in significantly less training steps that standard HRL methods (`HRL(seq)` and `HRL(pop)`).

The experimental results were positive and show that the approaches that restrict which options can be used based on the actions presented by the plan produced good policies after very few training episodes. In these discrete domains, the `naive` approach of directly following the high-level plan quickly reached a plateau in its performance; but even when compared to this, standard HRL methods needed up to an order of magnitude more training episodes to reach comparable results. The use of a partial-order plan resulted in significant improvements in the quality of the solutions found. For example, after only $10,000$ training episodes in the OfficeWorld domain, $\texttt{HRL}_\texttt{r}$found solutions that were approximately 3 steps removed from the best solutions found (normalized discounted reward of $\gamma^3 \approx 0.73$). With the same amount of training, the solutions found by `naive` were approximately 6 steps away from the purported optimum ($\gamma^6 \approx 0.53$). Without restricting the options, the solutions found after $10,000$ training episodes were over 12 steps longer than optimal ($\gamma^{12} \approx 0.28$). To achieve a performance similar to $\texttt{HRL}_\texttt{r}$, `HRL(pop)` needed over $30,000$ training episodes.

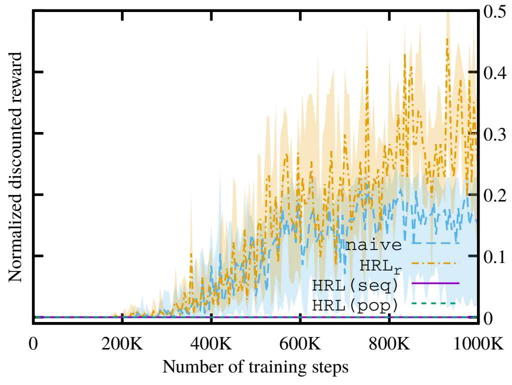
The effects of restricting the options and using partial-order plans were even more pronounced in the MineCraftWorld domain, where the environment and the tasks were much harder to learn. Indeed, after $100,000$ training episodes, $\texttt{HRL}_\texttt{r}$found solutions that were 3 steps away from optimal, on average, whereas `naive` found solutions that were 12 steps away. With this amount of training, all the policies learned with the unrestricted methods (`HRL(seq)` and `HRL(pop)`) produced solutions that were at least 35 steps longer than the best solutions found by other methods. In most cases,

[2]Available online at `http://www.fast-downward.org/`

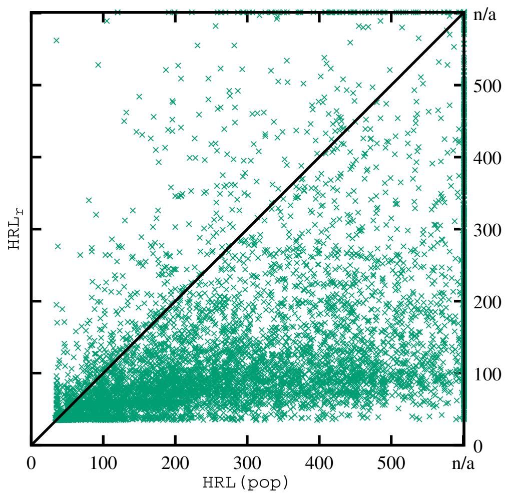
[3]With implementation available at `https://bitbucket.org/haz/pop-gen/`

they produced policies that could not actually reach the goal at all. To find solutions similar to those obtained with $\texttt{HRL}_\texttt{r}$and $100,000$ training episodes, HRL(pop) needed almost $500,000$ training episodes.

The difference in performance was exacerbated even further in the experiments on the continuous case. Here, the tasks are extremely hard and the baseline methods usually only found severely suboptimal solutions for them. We display these results in Figure 4.



(a) Median (and median quintile, shaded) discounted reward normalized by best result obtained for each task across all experiments.



(b) Comparison of the lengths of the solutions generated by $\texttt{HRL}_\texttt{r}$ and HRL(pop) throughout all experiments. $\texttt{HRL}_\texttt{r}$ found shorter solutions in 86.7% of experiments.

Figure 4: Experimental performance obtained on previously unseen tasks in the FarmWorld.

Figure 4a shows the discounted rewards obtained by all approaches in this domain. Here, we evaluated the results every $1,000$ training episodes. Note that even after $1,000,000$ training episodes, the best approach only found solutions that were over 30 steps longer than optimal on average. The approaches that do not restrict the available options typically found solutions that were hundreds of steps longer than optimal or that could not reach the goal. In Figure 4b, we compare the number of steps taken by the agent before reaching the goal across all experiments ran using $\texttt{HRL}_\texttt{r}$ and HRL(pop). Here, every point represents a different experiment. Points below the diagonal are those in which $\texttt{HRL}_\texttt{r}$ reached the goal in less steps than HRL(pop). Points along the rightmost and topmost borders respectively represent experiments in which $\texttt{HRL}_\texttt{r}$ reached the goal and HRL(pop) did not, and vice versa. In the $8,000$ experiments ran in the FarmWorld domain, $\texttt{HRL}_\texttt{r}$ found a shorter solution than HRL(pop) in $6,936$ cases, which corresponds to 86.7% of the cases.

6 Conclusions and Future Work

To conclude, we believe that the automatic generation of goal-relevant options and ordering constraints over them—in conjunction with the ability to explicitly represent them in a structured reward function—is one of the key aspects that will enable RL systems to be both taskable and scalable. Moreover, we believe that symbolic planning and knowledge representation techniques can provide the ideal framework for generating such options and constraints. Planning formalisms and algorithms support a plethora of different domain and solution properties, such as temporally-extended goals [2], preferences [3], diverse plans [20, 22], temporal and numeric constraints [7], and many more. All of these could be used for representing rich reinforcement learning tasks, while preserving useful structure that can be exploited when learning policies.

References

[1] Jacob Andreas, Dan Klein, and Sergey Levine. "Modular Multitask Reinforcement Learning with Policy Sketches". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Vol. 70. PMLR. 2017, pp. 166–175.

[2] Fahiem Bacchus and Froduald Kabanza. "Planning for Temporally Extended Goals". In: *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*. 1996, pp. 1215–1222.

[3] Jorge A. Baier and Sheila A. McIlraith. "Planning with Preferences". In: *AI Magazine* 29.4 (2008), pp. 25–36.

[4] Peter Dayan and Geoffrey E. Hinton. "Feudal Reinforcement Learning". In: *Advances in Neural Information Processing Systems 5 (NIPS)*. 1992, pp. 271–278.

[5] Thomas G. Dietterich. "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition". In: *Journal of Artificial Intelligence Research* 13 (2000), pp. 227–303.

[6] Minh B. Do and Subbarao Kambhampati. "Improving Temporal Flexibility of Position Constrained Metric Temporal Plans". In: *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*. 2003, pp. 42–51.

[7] Maria Fox and Derek Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 61–124.

[8] Matthew Jon Grounds and Daniel Kudenko. "Combining Reinforcement Learning with Symbolic Planning". In: *Adaptive Agents and Multi-Agents Systems III (AAMAS III)*. Vol. 4865. LNCS. 2008, pp. 75–86.

[9] Marek Grześ and Daniel Kudenko. "Plan-based reward shaping for reinforcement learning". In: *Proceedings of the 4th IEEE International Conference on Intelligent Systems (IS)*. Vol. 2. 2008, 10-22–10-29.

[10] Malte Helmert. "The Fast Downward planning system". In: *Journal of Artificial Intelligence Research* 26 (2006), pp. 191–246.

[11] Steven James, Benjamin Rosman, and George Konidaris. "Learning to Plan with Portable Symbols". In: *Workshop on Planning and Learning (PAL@ICML/IJCAI/AAMAS)*. 2018.

[12] George Konidaris and Andrew G. Barto. "Building Portable Options: Skill Transfer in Reinforcement Learning". In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*. 2007, pp. 895–900.

[13] George Konidaris, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning". In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 215–289.

[14] Daoming Lyu et al. "SDRL: Interpretable and Data-efficient Deep Reinforcement Learning Leveraging Symbolic Planning". In: *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*. 2019.

[15] Drew McDermott. *PDDL—The Planning Domain Definition Language*. Tech. rep. Yale Center for Computational Vision and Control, 1998.

[16] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[17] Christian J. Muise, J. Christopher Beck, and Sheila A. McIlraith. "Optimal Partial-Order Plan Relaxation via MaxSAT". In: *Journal of Artificial Intelligence Research* 57 (2016), pp. 113–149.

[18] Ronald Parr and Stuart J. Russell. "Reinforcement Learning with Hierarchies of Machines". In: *Advances in Neural Information Processing Systems 10 (NIPS)*. 1997, pp. 1043–1049.

[19] Silvia Richter and Matthias Westphal. "The LAMA planner: Guiding cost-based anytime planning with landmarks". In: *Journal of Artificial Intelligence Research* 39 (2010), pp. 127–177.

[20] Mark Roberts, Adele E. Howe, and Indrajit Ray. "Evaluating Diversity in Classical Planning". In: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*. 2014, pp. 253–261.

[21] Satinder P. Singh. "Reinforcement Learning with a Hierarchy of Abstract Models". In: *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI)*. 1992, pp. 202–207.

[22] Shirin Sohrabi et al. "Finding Diverse High-Quality Plans for Hypothesis Generation". In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*. 2016, pp. 1581–1582.

[23] Richard S. Sutton, Doina Precup, and Satinder P. Singh. "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112.1-2 (1999), pp. 181–211.

[24] Sebastian Thrun and Anton Schwartz. "Finding Structure in Reinforcement Learning". In: *Advances in Neural Information Processing Systems 7 (NIPS)*. 1994, pp. 385–392.

[25] Rodrigo Toro Icarte et al. "Advice-Based Exploration in Model-Based Reinforcement Learning". In: *Canadian Conference on Artificial Intelligence*. 2018, pp. 72–83.

[26] Rodrigo Toro Icarte et al. "Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Vol. 80. PMLR. 2018, pp. 2112–2121.

[27] Christopher J. C. H. Watkins. "Learning from delayed rewards". PhD thesis. King's College, University of Cambridge, 1989.

[28] Fangkai Yang et al. "PEORL: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. 2018, pp. 4860–4866. DOI: `10.24963/ijcai.2018/675`. URL: `https://doi.org/10.24963/ijcai.2018/675`.

[29] Amy Zhang et al. "Composable Planning with Attributes". In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Vol. 80. PMLR. 2018, pp. 5837–5846.