Diversity-Driven Extensible Hierarchical Reinforcement Learning

Yuhang Song,[1] Jianyi Wang,[2,†] Thomas Lukasiewicz,[1] Zhenghua Xu,[1,3,*] Mai Xu[2]
[1]Department of Computer Science, University of Oxford, UK
[2]School of Electronic and Information Engineering, Beihang University, China
[3]State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology, China
[*]Corresponding author, email: xuzhenghua1987@gmail.com
[†]Co-first author

Abstract

Hierarchical reinforcement learning (HRL) has recently shown promising advances on speeding up learning, improving the exploration, and discovering intertask transferable skills. Most recent works focus on HRL with two levels, i.e., a master policy manipulates subpolicies, which in turn manipulate primitive actions. However, HRL with multiple levels is usually needed in many real-world scenarios, whose ultimate goals are highly abstract, while their actions are very primitive. Therefore, in this paper, we propose a diversity-driven extensible HRL (DEHRL), where an extensible and scalable framework is built and learned levelwise to realize HRL with multiple levels. DEHRL follows a popular assumption: diverse subpolicies are useful, i.e., subpolicies are believed to be more useful if they are more diverse. However, existing implementations of this diversity assumption usually have their own drawbacks, which makes them inapplicable to HRL with multiple levels. Consequently, we further propose a novel diversity-driven solution to achieve this assumption in DEHRL. Experimental studies evaluate DEHRL with nine baselines from four perspectives in two domains; the results show that DEHRL outperforms the state-of-the-art baselines in all four aspects.

Introduction

Hierarchical reinforcement learning (HRL) recombines sequences of basic actions to form subpolicies (Sutton, Precup, and Singh 1999; Parr and Russell 1998; Dietterich 2000). It can be used to speed up the learning (Bacon, Harb, and Precup 2017), improve the exploration to solve tasks with sparse *extrinsic rewards* (i.e., rewards generated by the environment) (Şimşek and Barto 2004), or learn meta-skills that can be transferred to new problems (Frans et al. 2018). Although most previous approaches to HRL require handcrafted subgoals to pretrain subpolicies (Heess et al. 2016) or require extrinsic rewards as supervisory signals (Vezhnevets et al. 2016), the recent ones seek to discover subpolicies without manual subgoals or pretraining. Most of them (Xu et al. 2018a; Bacon, Harb, and Precup 2017) are working in a top-down fashion, where a given agent first explores until it accomplishes a trajectory that reaches a positive extrinsic reward, and then tries to recombine the basic actions in the trajectory to build subpolicies.

However, such top-down solutions are not practical in situations with sparse extrinsic rewards and large action space, where extrinsic rewards are almost unreachable by exploration using primitive actions. Therefore, more recent works focus on discovering "useful" subpolicies in a bottom-up fashion (Lakshminarayanan et al. 2016; Kompella et al. 2017), which are capable of discovering subpolicies before reaching any extrinsic rewards. In addition, the bottom-up strategy can discover subpolicies that better facilitate learning for an unseen problem (Frans et al. 2018). This is also called *meta-reinforcement-learning* (Al-Shedivat et al. 2018), where subpolicies are shared across different tasks and thus called *meta subpolicies*.

However, none of the above methods shows the capability to build extensible HRL with multiple levels, i.e., building subpolicies upon subpolicies, which is usually needed in many real-world scenarios, whose *ultimate goals* are highly abstract, while their basic *actions* are very primitive. We take the game *OverCooked* in Figure 1 as an example. The ultimate goal of OverCooked is to let an *agent* fetch multiple *ingredients* (purple boxes) in a particular sequence according to a *to-pick list* (orange box), which is shuffled in every episode. However, the basic action of the agent is so primitive (thus called *primitive action* in the following) that it can just move one of its four legs towards one of the four directions at each step (marked by red arrows), and the agent's body can be moved only when all four legs are moved to the same direction. Consequently, although we can simplify the task by introducing subpolicies to learn to move the body towards different directions with four steps of primitive actions, it is still difficult to reach the ultimate goal, because the to-pick list changes in every episode.

Fortunately, this problem can be easily overcome if HRL has multiple levels: by defining the previous subpolicies as subpolicies at level 0, HRL with multiple levels can build subpolicies at level 1 to learn to fetch different ingredients based on subpolicies at level 0; obviously, a policy based on subpolicies at level 1 is capable to reach the ultimate goal more easily than that based on subpolicies at level 0. Motivated by the above observation, we propose Diversity-driven Extensible HRL (*DEHRL*), which is constructed and trained levelwise (i.e., each level shares the same structure and is trained with exactly the same algorithm) to make it extensible to build higher levels. DEHRL follows a widely adopted

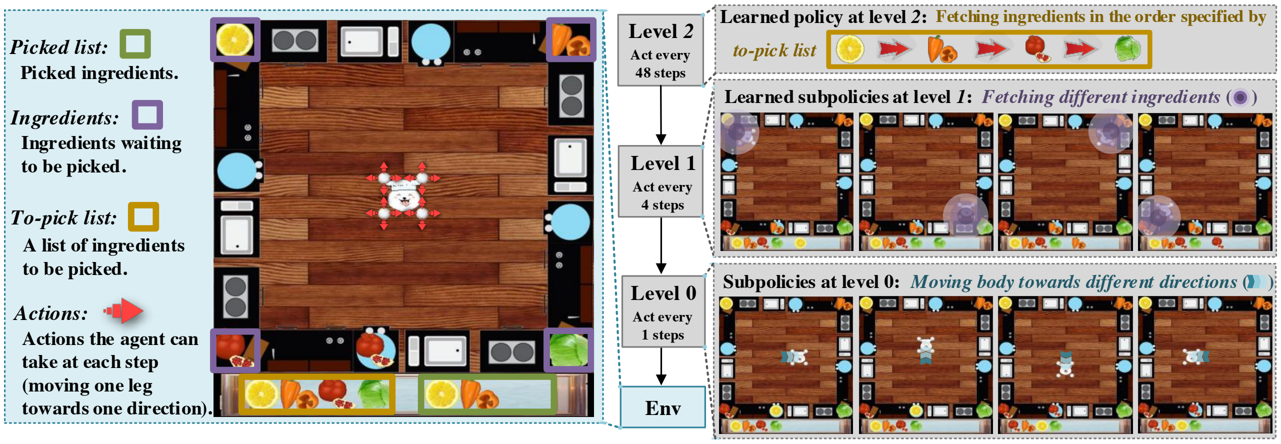

Picked list:
Picked ingredients.
Ingredients:
Ingredients waiting to be picked.
To-pick list:
A list of ingredients to be picked.
Actions:
Actions the agent can take at each step (moving one leg towards one direction).
Level 2
Act every 48 steps
Level 1
Act every 4 steps
Level 0
Act every 1 steps
Env
Learned policy at level 2: Fetching ingredients in the order specified by to-pick list
Learned subpolicies at level 1: Fetching different ingredients
Subpolicies at level 0: Moving body towards different directions

Figure 1: Playing OverCooked with HRL of three levels.

diversity assumption (Gregor, Rezende, and Wierstra 2016; Florensa, Duan, and Abbeel 2017): subpolicies are believed to be more useful if they are more diverse. Therefore, the objective of DEHRL at each level is to learn corresponding subpolicies that are as diverse as possible, thus called diversity-driven. The intuition of this diversity assumption is as follows: in real-world situations, when there is no specific stimulation, people tend to learn skills as diverse as possible to prepare for potential future needs; similarly, when there is no extrinsic reward, by learning subpolicies as diverse as possible, an agent will be able to explore its own capability as much as possible.

However, existing implementations of the diversity assumption usually have their own drawbacks, which make them inapplicable to HRL with multiple levels: (i) The implementation in (Daniel et al. 2016) works in a top-down fashion, making it impractical in situations with sparse extrinsic rewards. (ii) Although SAC-LSP in (Haarnoja et al. 2018) can also learn policy of multiple layers, whether it can operate different layers at different temporal scales is an open problem, making it also unable to solve tasks with sparse extrinsic rewards. (iii) The implementation in (Gregor, Rezende, and Wierstra 2016; Florensa, Duan, and Abbeel 2017) is not extensible to higher levels.

Consequently, we further propose a novel diversity-driven solution to achieve this diversity assumption in DEHRL. We first introduces a *predictor* at each level to dynamically predict the resulting state of each subpolicy. Then, the diversity assumption is achieved by giving higher *intrinsic rewards* (rewards generated by the agent) to subpolicies that result in more diverse states. Consequently, subpolicies converge to taking actions that result in most diverse states.

The contributions of this paper are summarized as follows. (i) We propose a diversity-driven extensible hierarchical reinforcement learning (DEHRL) approach, which, to our knowledge, is the first learning algorithm that is built and learned levelwise with verified scalability, so that HRL with multiple levels can be realized end-to-end without human-designed extrinsic rewards. (ii) We further propose a new diversity-driven solution to achieve the widely adopted diversity assumption in HRL with multiple levels. (iii) Experimental studies are conducted to compare DEHRL with nine baselines from four perspectives in two domains. The results show that: (a) DEHRL can discover useful subpolicies more effectively; (b) DEHRL can solve the sparse extrinsic reward problem more efficiently; (c) DEHRL can learn better intertask-transferable meta subpolicies; and (d) DEHRL has a good portability.

Diversity-Driven Extensible HRL

The structure of DEHRL is shown in Figure 2, where each level l contains a *policy* (denoted π^l), a *predictor* (denoted β^l), and an *Intrinsic Reward Function Module* (abbreviated as IRFM). The *policy* and *predictor* are two deep neural networks (i.e., parameterized functions) with π^l and β^l denoting their trainable parameters, while IRFM contains a set of unparameterized functions. For any two neighboring levels (e.g., level l and level $l-1$), there are three connections as shown in Figure 2: (i) the policy at the upper level π^l produces an action a_t^l, which is treated as an input for the policy at the lower level π^{l-1}; (ii) the predictor at the upper level β^l makes several predictions, which are passed to IRFM at the lower level $l-1$; (iii) using the predictions from the upper level l, IRFM at the lower level $l-1$ generates an intrinsic reward b_t^{l-1} to train the policy at the lower level π^{l-1}. The rest of this section introduces some important information of the proposed DEHRL framework and the diversity-driven solution; more details and theoretical interpretations of DEHRL can be found in (Song et al. 2018a).

Policy

As shown in Figure 2, the policies for different levels act at different frequencies, i.e., the policy π^l samples an action every T^l steps. Note that T^l is always an integer multiple of T^{l-1}, and T^0 always equals to 1, so the time complexity of the proposed framework does not grow linearly as the level goes higher. T^l for $l > 0$ are hyper-parameters. At level l, the policy π^l takes as input the current state s_t and the action from the upper level a_t^{l+1}, so that the output of π^l

is conditional to a_t^{l+1}. Note that $a_t^{l+1} \in \mathbb{A}^{l+1}$, where $\mathbb{A}^{l+1}$ is the output action space of π^{l+1}. Thus, $\mathbb{A}^0$ should be set to the action space of the *environment* for the policy π^0 directly taking actions on the environment, while $\mathbb{A}^l$ of $l > 0$ are hyper-parameters. The policy takes as input both s_t and a_t^{l+1} to integrate multiple subpolicies into one model; a similar idea is presented in (Florensa, Duan, and Abbeel 2017). The detailed network structure of the policy π^l is presented in (Song et al. 2018a). Then, the policy π^l produces the action $a_t^l \in \mathbb{A}^l$ by sampling from a parameterized categorical distribution:

$$a_t^l = \pi^l(s_t, a_t^{l+1}). \tag{1}$$

The reward to train π^l combines the extrinsic reward from the environment r_t^{env} and the intrinsic reward b_t^l generated from IRFM at level l. When facing games with very sparse extrinsic rewards, where r_t^{env} is absent most of the time, b_t^l will guide the policy at this level π^l to learn diverse subpolicies, so that the upper level policy π^{l+1} may reach the sparse positive extrinsic reward more easily. The policy π^l is trained with the PPO algorithm (Schulman et al. 2017), but our framework does not restrict the policy training algorithm to use. The following denotes the loss of training the policy π^l:

$$L_{\text{policy}}^l = \text{PPO}\left(a_t^l, (r_t^{\text{env}} + \lambda b_t^l)|\pi^l\right), \tag{2}$$

where π^l means that the gradients of this loss are only passed to the parameters in π^l, and λ is a hyper-parameter set to 1 all the time (the section on IRFM below will introduce a normalization of the intrinsic reward b_t^l, making λ free from careful tuning).

Predictor

As shown in Figure 2, the predictor at level l (i.e., β^l) takes as input the current state s_t and the action taken by the policy at level l (i.e., a_t^l) as a one-hot vector. The integration of s_t and a_t^l is accomplished by approximated multiplicative interaction (Oh et al. 2015), so that any predictions made by the predictor β^l is conditioned on the action input of a_t^l. The predictor makes two predictions, denoted $\hat{s}_{t+T^l}$ and $\hat{b}_t^{l-1}$, respectively. Thus, the forward function of β^l is:

$$\hat{s}_{t+T^l}, \hat{b}_t^{l-1} = \beta^l(s_t, a_t^l). \tag{3}$$

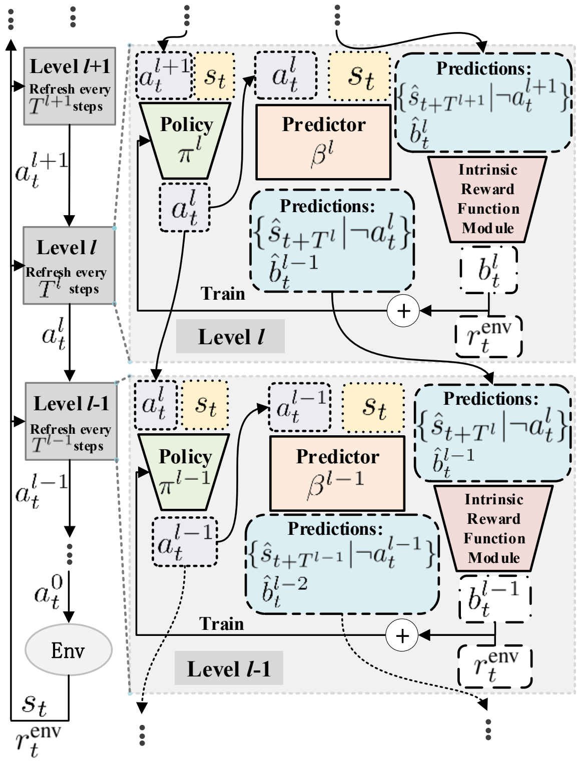
The detailed network structure of the predictor β^l is given in (Song et al. 2018a). The first prediction $\hat{s}_{t+T^l}$ in (3) is trained to predict the state after T^l steps with following loss function:

$$L_{\text{transition}}^l = \text{MSE}(s_{t+T^l}, \hat{s}_{t+T^l}|\beta^l), \tag{4}$$

where MSE is the mean square error, and β^l indicates that the gradients of this loss are only passed to the parameters in β^l. The second prediction $\hat{b}_t^{l-1}$ in (3) is trained to approximate the intrinsic reward at the lower level b_t^{l-1}, with the loss function

$$L_{\text{intrinsic reward}}^l = \text{MSE}(b_t^{l-1}, \hat{b}_t^{l-1}|\beta^l), \tag{5}$$

where β^l means that the gradients of this loss are only passed to the parameters in β^l. The next section about IRFM will



Level l+1
Refresh every T^{l+1} steps
a_t^{l+1}
Level l
Refresh every T^l steps
a_t^l
Level l-1
Refresh every T^{l-1} steps
a_t^{l-1}
a_t^0
Env
s_t
r_t^{env}
a_t^{l+1}
s_t
a_t^l
s_t
Predictions: $\{\hat{s}_{t+T^{l+1}}|\neg a_t^{l+1}\}$ $\hat{b}_t^l$
Policy π^l
Predictor β^l
Intrinsic Reward Function Module
a_t^l
Predictions: $\{\hat{s}_{t+T^l}|\neg a_t^l\}$ $\hat{b}_t^{l-1}$
b_t^l
Train
r_t^{env}
Level l
a_t^l
s_t
a_t^{l-1}
s_t
Predictions: $\{\hat{s}_{t+T^l}|\neg a_t^l\}$ $\hat{b}_t^{l-1}$
Policy π^{l-1}
Predictor β^{l-1}
Intrinsic Reward Function Module
a_t^{l-1}
Predictions: $\{\hat{s}_{t+T^{l-1}}|\neg a_t^{l-1}\}$ $\hat{b}_t^{l-2}$
b_t^{l-1}
Train
r_t^{env}
Level l-1

Figure 2: The framework of DEHRL.

show that the intrinsic reward b_t^{l-1} is also related to the action a_t^{l-1} sampled according to the policy at the lower level π^{l-1}. Since a_t^{l-1} is not fed into the predictor β^l, the intrinsic reward $\hat{b}_t^{l-1}$ is actually an estimation of the expectation of b_t^{l-1} under the current π^{l-1}:

$$\hat{b}_t^{l-1} = \mathbb{E}_{\pi^{l-1}}\{b_t^{l-1}\}. \tag{6}$$

The above two predictions will be used in IRFM, described in the following section.

The predictor is active at the same frequency as that of the policy. Each time the predictor β^l is active, it produces several predictions feeding to IRFM at level $l-1$, including $\hat{b}_t^{l-1}$ and $\{\hat{s}_{t+T^l}|\neg a_t^l\}$, where $\{\hat{s}_{t+T^l}|\neg a_t^l\}$ denotes a set of predictions when feeding the predictor β^l with actions other than a_t^l. In practice, the predictor is affected by the distribution of the sampled data from the online policy; DEHRL thus adds an entropy regularization loss in the training to encourage the policy to be uniform over its action space.

Intrinsic Reward Function Module (IRFM)

As shown in Figure 2, taking as input $\hat{b}_t^{l-1}$ and $\{\hat{s}_{t+T^l}|\neg a_t^l\}$, IRFM produces the intrinsic reward b_t^{l-1}, which is used to train the policy π^{l-1}, as described in the policy section. The design of IRFM is motivated as follows: (i) If the currently selected subpolicy $\pi^{l-1}(s_t, a_t^l)$ for the upper-level action a_t^l differs from the subpolicies for other actions (i.e., $\{\pi^{l-1}(s_t, \neg a_t^l)\}$), then the intrinsic reward b_t^{l-1} should be high; (ii) The above difference can be measured

Table 1: The settings of DEHRL.

$\mathbb{A}^0$	$\mathbb{A}^1$	$\mathbb{A}^2$	T^0	T^1	T^2
16	5	5	1	1*4	1*4*12

via the distance between the states resulting from the subpolicy $\pi^{l-1}(s_t, a_t^l)$ and subpolicies $\{\pi^{l-1}(s_t, \neg a_t^l)\}$. Note that since a_t^l is selected every T^l steps, these resulting states are the ones at $t + T^l$.

In the above motivation, the resulting state of the subpolicy $\pi^{l-1}(s_t, a_t^l)$ is the real state environment returned after T^l steps (i.e., s_{t+T^l}), while the resulting states of the subpolicies $\{\pi^{l-1}(s_t, \neg a_t^l)\}$ have been predicted by the predictor at the upper level l (i.e., $\{\hat{s}_{t+T^l} | \neg a_t^l\}$), as described in the last section. Thus, the intrinsic reward b_t^{l-1} is computed as follows:

$$b_t^{l-1} = \sum_{s \in \{\hat{s}_{t+T^l} | \neg a_t^l\}} D(s_{t+T^l}, s), \tag{7}$$

where D is the distance chosen to measure the distance between states. In practice, we combine the L1 distance and the distance between the center of mass of the states, to obtain information on color changes as well as objects moving. A more advanced way to measure the above distance is to match features in states and to measure the movements of the matched features, or to integrate the inverse model in (Pathak et al. 2017) to capture the action-related feature changes. However, the above advanced ways are not investigated here, due to the scope of the paper. Equation (7) gives a high intrinsic reward, if s_{t+T^l} is far from $\{\hat{s}_{t+T^l} | \neg a_t^l\}$ overall. In practice, we find that punishing s_{t+T^l} from being too close to the one state in $\{\hat{s}_{t+T^l} | \neg a_t^l\}$ that is closest to s_{t+T^l} is a much better choice. Thus, we replace the sum in (7) with the minimum, and find that it consistently gives the best intrinsic reward estimation.

Estimating the intrinsic reward with distances of high dimensional states comes with the problem that the changes in distance that we want the intrinsic reward to capture is extremely small, compared to the mean of the distances. Thus, we use the estimation of the expectation of the intrinsic reward b_t^{l-1} (i.e., $\hat{b}_t^{l-1}$ described in last section) to normalize b_t^{l-1}:

$$b_t^{l-1} \leftarrow b_t^{l-1} - \hat{b}_t^{l-1}. \tag{8}$$

In practice, this normalization gives a stable algorithm without need to tune λ according to the distance that we choose or the convergence status of the predictor at the upper level.

Experiments

We conduct experiments to evaluate DEHRL with nine baselines from four perspectives based on two domains, OverCooked (shown in Figure 1) and Minecraft. The important hyper-parameters of DEHRL are summarized in Table 1, while other details (e.g., neural network architectures and hyper-parameters in the policy training algorithm) are provided in (Song et al. 2018a). Easy-to-run codes have been released to further clarify the details and facilitate future research[1], where evaluations and visualizations on more domains, such as Montezuma's Revenge and PyBullet (an open source alternative of MuJoCo), can also be found.

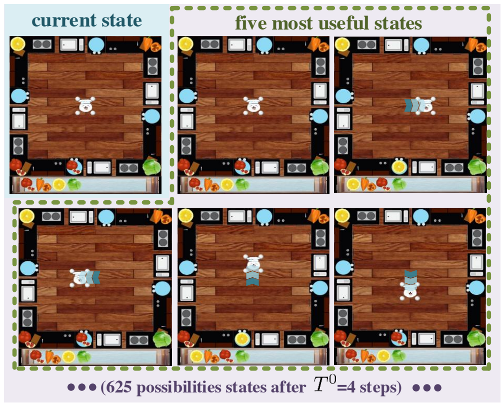


Figure 3: State examples of Overcooked.

Subpolicy Discovery

We first evaluate the performance of DEHRL in discovering diverse subpolicies in OverCooked. As shown in Figure 1, an agent in OverCooked can move one of its four legs towards one of four directions at each step, so its action space is 16. Only after all four legs are moved towards the same direction, the body of this agent can be moved towards this direction. There are four different ingredients at the corners of the kitchen (purple boxes). An ingredient is automatically picked up when the agent reaches it. There is a list of ingredients that the chief needs to pick in sequence to complete a dish (orange box), called to-pick list.

Without Extrinsic Rewards. We first aim to discover useful subpolicies without extrinsic rewards. Since there are four legs, and every leg of an agent has five possible states (staying or moving towards four directions), there are totally $5^4 = 625$ possible states for every $T^0 = 4$ steps. As shown in Figure 3, five of them are the most useful states (i.e., the ones that are most diverse to each other), whose four legs have the same state, making the body of the chief move towards four directions or stay still.

Consequently, a good implementation of the diversity assumption should be able to learn subpolicies at level 0 that can result in the five most useful states (called *five useful subpolicies*) efficiently and comprehensively. Therefore, given $\mathbb{A}^1 = 5$ (i.e., discovering only five subpolicies), and the number of steps being 10 millions, the five subpolicies learned by DEHRL at level 0 are exactly the five useful subpolicies. Furthermore, *SNN* (Florensa, Duan, and Abbeel 2017) is a state-of-the-art implementation of the diversity assumption, which is thus tested as a baseline under the same setting. However, only one of the five useful subpolicies is discovered by SNN. We then repeat experiments ten times with different training seeds, and the results are the same.

[1]https://github.com/YuhangSong/DEHRL

Table 2: The different settings of extrinsic rewards in OverCooked.

reward-level	*goal-type: any* (easy)	*goal-type: fix* (medium)	*goal-type: random* (hard)
1 (easy)	Get any ingredient.	Get a particular ingredient.	Get the first ingredient shown in the shuffling* to-pick list.
2 (hard)	Get 4 ingredients in any order.	Get 4 ingredients in a particular order.	Get 4 ingredients in order according to the shuffling* to-pick list.

* The to-pick list is shuffled every episode.

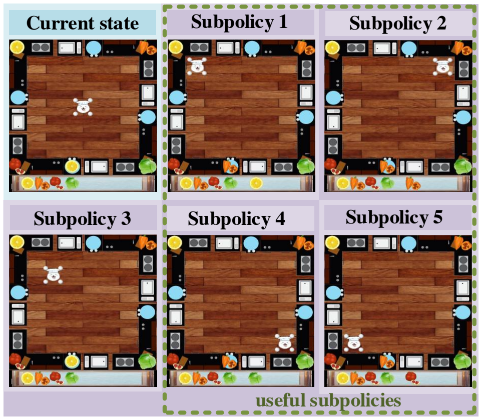


Figure 4: Subpolicies learned at level 1 in DEHRL.

Furthermore, we loose the restriction by setting $\mathbb{A}^1 = 20$ (i.e., discovering 20 subpolicies) and the number of steps is 20 millions. With no surprise, the five useful subpolicies are always included in the 20 subpolicies discovered by DEHRL; however, the 20 subpolicies discovered by SNN still contain only one useful subpolicy.

The superior performance of DEHRL comes from the proposed new diversity-driven solution, which gives higher intrinsic rewards to subpolicies that result in more diverse states; consequently, subpolicies in DEHRL converge to taking actions that result in most diverse states. And the failure of SNN may be because the objective of SNN is to maximize mutual information, so it only guarantees to discover subpolicies resulting in different states, but these different states are not guaranteed to be most diverse to each other. Similar failures are found in other state-of-the-art solutions, e.g., (Gregor, Rezende, and Wierstra 2016); we thus omit the analysis of these methods due to space limit.

As for finding useful subpolicies at higher levels, due to the failures at level 0, none of the state-of-the art solutions can generate useful subpolicies at higher levels. However, useful subpolices can still be learned by DEHRL at higher levels. Figure 4 visualizes five subpolicies learned by DEHRL at level 1, where four of them (marked by a green dash-line box) result in getting ingredients at four different corners, which are the useful subpolicies at level 1.

With Extrinsic Rewards. We further compare DEHRL with three state-of-the-art methods, *option-critic* (Bacon, Harb, and Precup 2017), *FeUdal* (Vezhnevets et al. 2017), and *DIAYN* (Eysenbach et al. 2018), when extrinsic rewards are given. As shown in Table 2, six different extrinsic reward settings with different difficulties are given to OverCooked.

Table 3: Final performance score of DEHRL and baselines on OverCooked with six different extrinsic reward settings.

reward-level / *goal-type*	*1* / *any*	*1* / *fix*	*1* / *random*	*2* / *any*	*2* / *fix*	*2* / *random*
DEHRL	**1.00**	**1.00**	**1.00**	**0.95**	**0.93**	**0.81**
Option-critic	**1.00**	**1.00**	0.00	0.00	0.00	0.00
DIAYN	**1.00**	**1.00**	0.83	0.42	0.10	0.00
FeUdal	**1.00**	**1.00**	0.93	0.00	0.00	0.00
PPO	0.98	0.97	0.56	0.00	0.00	0.00
State Novelty	**1.00**	0.96	0.95	0.00	0.00	0.00
Transition Novelty	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00

To measure the performance quantitatively, two metrics, *final performance score* and *learning speed score*, based on reward per episode, are imported from (Schulman et al. 2017). Generally, the higher the reward per episode, the better the solution. Specifically, the final performance score averages the reward per episode over the last 100 episodes of training to measure the performance at final stages; while the learning speed score averages the extrinsic reward per episode over the entire training period to quantify the learning efficiency.

Table 3 show results of the final performance scores. As shown in Table 3, DEHRL can solve the problems in all six settings. However, option-critic only solves the two easier ones; the failure of option-critic is because the extrinsic reward gets more sparse in the last four harder cases. In addition, FeUdal fails when it is extended to 3 levels, because its key idea "transition policy gradient" does not work well for multi-level structures, making it hard to converge for *reward-levels* $= 2$. Similarly, DIAYN also fails when the *reward-levels* $= 2$, showing that only achieving diversity over primitive actions is not enough. Consequently, we state that DEHRL can also achieve better performances than the state-of-the-art baselines when extrinsic rewards are given.

Solving the Sparse Extrinsic Reward Problem

Then, we further compare DEHRL with two state-of-the-art methods, *state novelty* (Şimşek and Barto 2004) and *transition novelty* (Pathak et al. 2017), towards improving exploration. In addition, as our framework is based on the *PPO* algorithm (Schulman et al. 2017), we include PPO as a baseline as well. The performances based on the final performance scores are also shown in Table 3. The results in Ta-

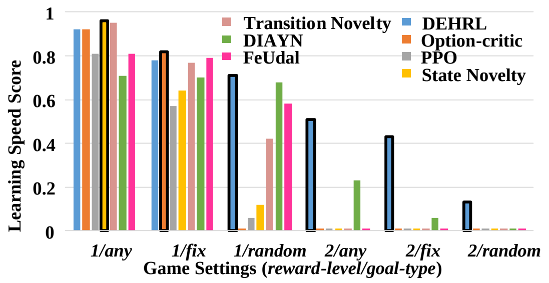


Figure 5: Learning speed score of DEHRL and the baselines on OverCooked with six different extrinsic reward settings.

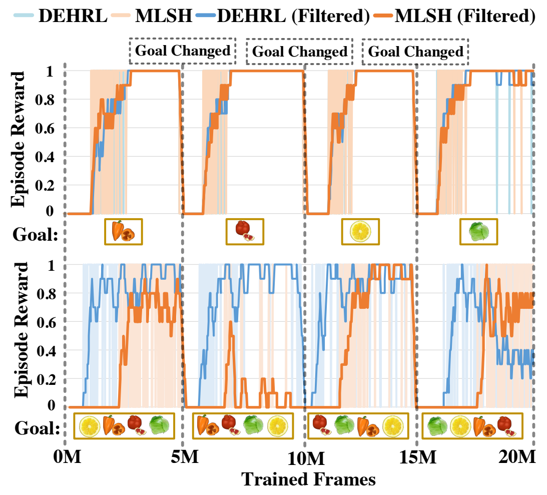


Figure 6: Meta HRL performances of DEHRL and MLSH in OverCooked.

ble 3 show that, these three baselines are all able to solve the task in the first three settings but all fail in the last three settings. This thus demonstrates that DEHRL can resolve the sparse extrinsic reward problem more effectively.

Finally, Figure 5 shows the learning efficiency in the learning speed score. We note that although DEHRL is much more complicated than the baselines, it still maintains very good learning efficiency in easy tasks: its learning speed scores are very close to the best of the baselines in the first two easier tasks. Furthermore, the learning efficiency of DEHRL greatly outperforms all baselines in the harder tasks. This asserts that, besides effectiveness, DEHRL also achieves superior learning efficiency. Also, efficiency measured by wall-clock time can be found in (Song et al. 2018a).

Meta HRL

HRL has recently shown a promising ability to learn meta-subpolicies to better facilitate adaptive behaviors for new problems (Solway et al. 2014; Frans et al. 2018). We thus further compare DEHRL with the state-of-the-art framework, MLSH (Frans et al. 2018), to investigate its performance in meta learning.

We first set *reward-level=1* and *goal-type=random* in OverCooked. The episode extrinsic rewards of both DEHRL and MLSH are plotted in Figure 6 (upper part), which always drop when the goal is changed, because the top-level hierarchies of both methods are re-initialized. Obviously, the increase speeds of the episode extrinsic rewards after each reset can measure the performances of methods in facilitating adaptive behaviors for a new goal (i.e., learning meta-subpolicies). Consequently, we find that DEHRL and MLSH have a similar meta-learning performance under this setting.

Then, we repeat the above experiment with *reward-level=2*, and the results are shown in Figure 6 (lower part). We find that DEHRL achieves a better meta-learning than MLSH in this setting. This is because DEHRL can learn the subpolicies at level 1 to fetch four different ingredients, while MLSH can only learn subpolicies to move towards four different directions. Obviously, subpolicies learned at level 1 are better intertask-transferable, which make DEHRL resolve the new goal more easily and quickly than MLSH.

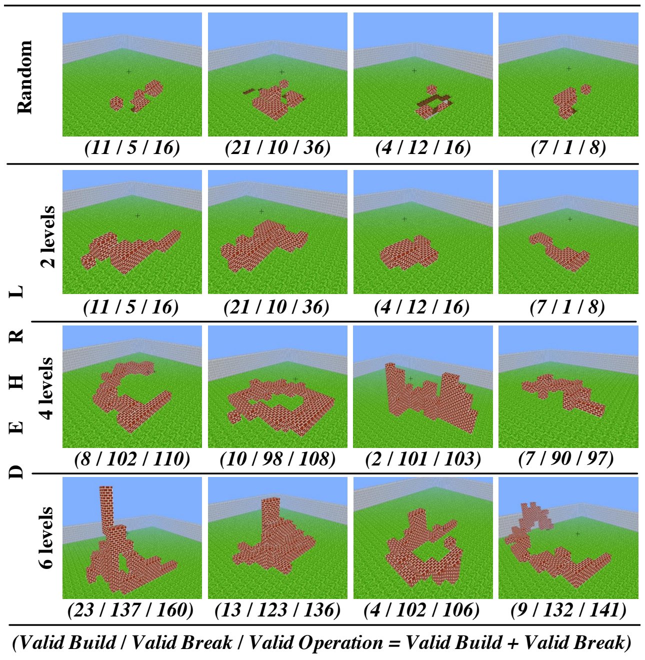


Figure 7: Worlds built by playing Minecraft without extrinstic reward.

Application of DEHRL in Minecraft

To show the portability of DEHRL, we further apply DEHRL in a popular video game called *Minecraft*, where the agent has much freedom to act and build, and it has a 1st-person view via raw pixel input. At the beginning of each episode, the world is empty except one layer of *GRASS* blocks that can be broken. We allow the agent to play 1000 steps in each episode; then the world is reset. At each step, ten actions are available: moving towards four directions, rotating the view towards four directions, break/build a block and jump. More detailed settings and visualizations are in (Song et al. 2018a).

Since the typical use of DEHRL is based on the intrinsic reward only, the existing work (Tessler et al. 2017) that re-

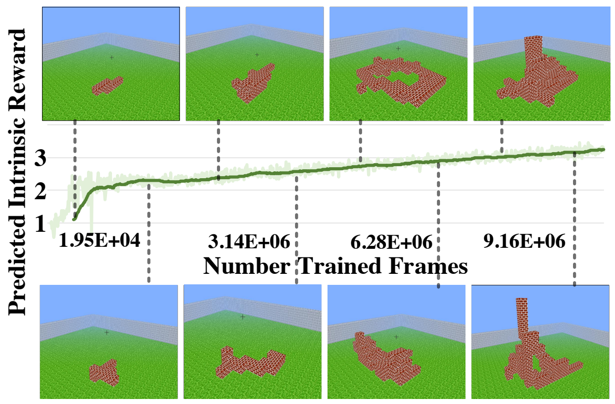


Figure 8: Predicted intrinsic reward.

quires human-designed extrinsic reward signals to train subpolicies is not appropriate to be used as the baseline. Consequently, we first construct three DEHRL models with three different numbers of levels, and then compare their performances in Minecraft to a framework with a random policy. Figure 7 shows the building results, where we measure the performance by world complexity. As shown in Figure 7, according to our intuitive feeling, DEHRL models obviously build more complex worlds than the random policy. Furthermore, with the increase of the number of levels, DEHRL tends to build more complex worlds.

Then, we further use a metric, *Valid Operations*, to quantify the complexity of the worlds, which is defined as:

$$Valid\ Operations = Valid\ Breaks + Valid\ Builds,$$

where *Valid Builds* is the number of blocks that have been built and not broken at the end of an episode; *Valid Breaks* is the number of blocks that are originally in the world that have been broken. Consequently, blocks that are built but broken later will not be counted into *Valid Build* or *Valid Break*. As in Figure 7, the quantitative results are consistent with our previous intuitive feelings. Here, although the diversity objective is defined under a 1st-person view, the diversity score is measured under a 3rd-person view; it is to follow an intuition that although human players have a 1st-person view in Minecraft, their underlying intrinsic rewards motivate them to build complex 3rd-person view structures.

Finally, as the predicted intrinsic reward ($\hat{b}_t^l$) is an indication of the diversity of current subpolicies, we plot in Figure 8 the averaged $\hat{b}_t^l$ over all levels and visualize the world built by DEHRL at different points to illustrate relationship between $\hat{b}_t^l$ and the built world. Minecraft with specific goals has also been investigated and included in (Song et al. 2018a).

Related Work

Discovering diverse subpolicies in HRL. The diversity assumption is prevailing in the recent works of option discovery. HiREPS (Daniel et al. 2016) is a popular approach, but it works in a top-down fashion. SNN (Florensa, Duan, and Abbeel 2017) is designed to handle sparse extrinsic reward tasks, which achieves this diversity assumption explicitly by information-maximizing statistics. Besides, it is promising to apply SNN in HRL of multiple levels. However, SNN suffers from various failures when the possible future states are enormous, making it impractical on domains with a large action space and unable to further learn higher-level subpolicies. Similar failure cases are observed in (Gregor, Rezende, and Wierstra 2016).

Extensible HRL. Recently, there has been some attempts in increasing the levels of HRL. Such works include MAXQ (Dietterich 2000), which requires completely searching the subtree of each subtask, leading to high computational costs. In contrast, AMDP (Gopalan et al. 2017) explores only the relevant branches. However, AMDP concentrates on the planning problem. Deeper levels are also supported in (Silver and Ciosek 2012), but its scalability is not clear. DDO (Fox et al. 2017) and DDCO (Krishnan et al. 2017) discover higher-level subpolicies from demonstration trajectories. However, our work focuses on learning those purely end-to-end without human-designed extrinsic reward. Other works (Rasmussen, Voelker, and Eliasmith 2017; Song et al. 2018b) also involve a modular structure that supports deeper-level HRL. However, there is no guarantee or verification on whether the structure can learn useful or diverse subpolicies at different temporal scales.

Meta HRL. Neuroscience research (Solway et al. 2014) proposes that the optimal hierarchy is the one that best facilitates an adaptive behavior in the face of new problems. Its idea is accomplished with a verified scalability in MLSH (Frans et al. 2018), where meta HRL is proposed. However, MLSH keeps reinitializing the policy at the top level, once the environment resets the goal. This brings several drawbacks, such as requiring auxiliary information from the environment about when the goal has been changed. In contrast, our method does not introduce such a restriction. Regardless of the difference, we compare with MLSH in our experiments under the settings of MLSH, where auxiliary information on goal resetting is provided. As such, the meta HRL ability of our approach is investigated.

Improved exploration with predictive models. Since we introduce the transition model to generate intrinsic rewards, our method is also related to RL improvements with predictive models, typically introducing sample models (Fu, Co-Reyes, and Levine 2017), generative models (Song et al. 2017), or deterministic models (Pathak et al. 2017) as transition models to predict future states. However, the transition model in our DEHRL is introduced to encourage developing diverse subpolicies, while those in the above works are introduced to improve the exploration. Our method is compared with the above state novelty (Şimşek and Barto 2004) and transition novelty (Pathak et al. 2017) in our experiments.

Summary and Outlook

We have proposed DEHRL towards building extensible HRL that learns useful subpolicies over multiple levels effectively and efficiently. One interesting future research work is to develop algorithms that generate or dynamically adjust the settings of T^l and $\mathbb{A}^l$. Furthermore, measuring the distance between states is another future research direction, where better representations of states may lead to improvements. Finally, DEHRL may be applied to visual tasks (Xu et al. 2018b) to achieve promising solution with diverse representations and mixed reward functions.

Acknowledgments. This work was supported by China Scholarship Council under the State Scholarship Fund, by the Alan Turing Institute under the UK EPSRC grant EP/N510129/1, and by Hebei University of Technology under the Yuanguang Scholar Plan.

References

Al-Shedivat, M.; Bansal, T.; Burda, Y.; Sutskever, I.; Mordatch, I.; and Abbeel, P. 2018. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *ICLR*.

Bacon, P.-L.; Harb, J.; and Precup, D. 2017. The option-critic architecture. In *AAAI*.

Daniel, C.; Neumann, G.; Kroemer, O.; and Peters, J. 2016. Hierarchical relative entropy policy search. *JMLR*.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *JAIR*.

Eysenbach, B.; Gupta, A.; Ibarz, J.; and Levine, S. 2018. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*.

Florensa, C.; Duan, Y.; and Abbeel, P. 2017. Stochastic neural networks for hierarchical reinforcement learning. In *ICLR*.

Fox, R.; Krishnan, S.; Stoica, I.; and Goldberg, K. 2017. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*.

Frans, K.; Ho, J.; Chen, X.; Abbeel, P.; and Schulman, J. 2018. Meta learning shared hierarchies. In *ICLR*.

Fu, J.; Co-Reyes, J.; and Levine, S. 2017. EX2: Exploration with exemplar models for deep reinforcement learning. In *NIPS*.

Gopalan, N.; desJardins, M.; Littman, M. L.; MacGlashan, J.; Squire, S.; Tellex, S.; Winder, J.; and Wong, L. L. 2017. Planning with abstract markov decision processes. In *ICAPS*.

Gregor, K.; Rezende, D. J.; and Wierstra, D. 2016. Variational intrinsic control. In *ICLR Workshop*.

Haarnoja, T.; Hartikainen, K.; Abbeel, P.; and Levine, S. 2018. Latent space policies for hierarchical reinforcement learning. In *ICML*.

Heess, N.; Wayne, G.; Tassa, Y.; Lillicrap, T.; Riedmiller, M.; and Silver, D. 2016. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*.

Kompella, V. R.; Stollenga, M.; Luciw, M.; and Schmidhuber, J. 2017. Continual curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. *AIJ*.

Krishnan, S.; Fox, R.; Stoica, I.; and Goldberg, K. 2017. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *CoRL*.

Lakshminarayanan, A. S.; Krishnamurthy, R.; Kumar, P.; and Ravindran, B. 2016. Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv preprint arXiv:1605.05359*.

Oh, J.; Guo, X.; Lee, H.; Lewis, R. L.; and Singh, S. 2015. Action-conditional video prediction using deep networks in atari games. In *NIPS*.

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

Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. In *ICML*.

Rasmussen, D.; Voelker, A.; and Eliasmith, C. 2017. A neural model of hierarchical reinforcement learning. *PLOS ONE*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., and Ciosek, K. 2012. Compositional planning using optimal option models. *arXiv preprint arXiv:1206.6473*.

Şimşek, Ö., and Barto, A. G. 2004. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *ICML*.

Solway, A.; Diuk, C.; Córdova, N.; Yee, D.; Barto, A. G.; Niv, Y.; and Botvinick, M. M. 2014. Optimal behavioral hierarchy. *PLOS Comput. Biol.*

Song, Y.; Grimm, C.; Wang, X.; and Littman, M. L. 2017. Learning approximate stochastic transition models. *arXiv preprint arXiv:1710.09718*.

Song, Y.; Wang, J.; Lukasiewicz, T.; Xu, Z.; and Xu, M. 2018a. Diversity-driven extensible hierarchical reinforcement learning. *arXiv preprint arXiv:1811.04324*.

Song, Y.; Xu, M.; Zhang, S.; and Huo, L. 2018b. Generalization tower network: A novel deep neural network architecture for multi-task learning. In *NIPS Workshop*.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *AIJ*.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D. J.; and Mannor, S. 2017. A deep hierarchical approach to lifelong learning in Minecraft. In *AAAI*.

Vezhnevets, A.; Mnih, V.; Osindero, S.; Graves, A.; Vinyals, O.; Agapiou, J.; et al. 2016. Strategic attentive writer for learning macro-actions. In *NIPS*.

Vezhnevets, A. S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; and Kavukcuoglu, K. 2017. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*.

Xu, D.; Nair, S.; Zhu, Y.; Gao, J.; Garg, A.; Fei-Fei, L.; and Savarese, S. 2018a. Neural task programming: Learning to generalize across hierarchical tasks. In *ICRA*.

Xu, M.; Song, Y.; Wang, J.; Qiao, M.; Huo, L.; and Wang, Z. 2018b. Predicting head movement in panoramic video: A deep reinforcement learning approach. *TPAMI*.