Guiding Task Learning by Hierarchical RL with an Experience Replay Mechanism Through Reward Machines

Jinmiao Cong, Yang Liu, and Chuanjuan Liu(✉)

School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China
cjm111@mail.dlut.edu.cn, {ly,chanjuanliu}@dlut.edu.cn

Abstract. Recently, reinforcement learning (RL) has made great progress in theory and application. Whereas, challenges remain in RL, such as low sample utilization and difficulty in designing suitable reward functions. Therefore, this paper focuses on optimizing the structure of the reward function and improving sample utilization. We propose a hierarchical reinforcement learning (HRL) algorithm based on the options framework, which incorporates a segmented reward mechanism and an experience replay mechanism. The reward mechanism can help the agent grasp the reward function's internal structure. The experience replay mechanism includes a buffer for storing typical experiences and a particular buffer for storing the special state experiences of the agent accessing the subtasks, which are conducive to training. We conducted single-task and multitask tests in multiple environments. Experimental results demonstrate that our algorithm has a better performance than baseline algorithms.

Keywords: HRL · Reward Mechanism · Experience Replay Mechanism

1 Introduction

To address low sample utilization and slow learning rate of RL [7] in large-scale problems, scholars have introduced the HRL algorithm, which decomposes intricate problems into subproblems, such as HAMs, MAXQ, and Options [5]. However, owing to the hierarchical structure, these methods may not guarantee convergence to the optimal strategy. To overcome this limitation, researchers leveraged the prior knowledge of the Reward Machines (RM) [4] to mitigate the problem of converging to suboptimal solutions. The RM has received widespread research attention, including LSRM [8], CRM [2], and HRM [2]. These works use RM to decompose tasks and to output the combination of reward functions under different conditions. In spite of solving some tasks in the environments, they are still prone to issues such as slow learning speed and poor performance.

F. Liu et al. (Eds.): PRICAI 2023, LNAI 14325, pp. 164–170, 2024.
https://doi.org/10.1007/978-981-99-7019-3_17

Our Contribution. In order to improve learning speed and problem-solving ability, this work proposes an algorithm, dubbed Hierarchy based on Options with an Experience Replay Mechanism (HOERM). HOERM adopts an option-based hierarchical structure, which is relevant to HRM [2]. However, we designed a well-structured reward mechanism and included an experience replay mechanism. We refined the segmented reward mechanism by using the value iteration method to calculate the potential values of each state in the RM (See Sect. 2 for details). The difference in potential values between states can serve as intermediate rewards. Therefore, the agent also receives appropriate rewards during the transition of intermediate states in the RM. Besides, we designed an experience replay mechanism that includes two replay buffers. $ReplayBuffer_1$ is a typical buffer that normally stores various experiences during the learning process. $ReplayBuffer_2$ is a special buffer that specifically stores experiences when the agent accesses task-specific states. We effectively utilized these experiences to help the agent learn the strategy and complete tasks faster. Finally, we proved the effectiveness of the proposed mechanism through experiments.

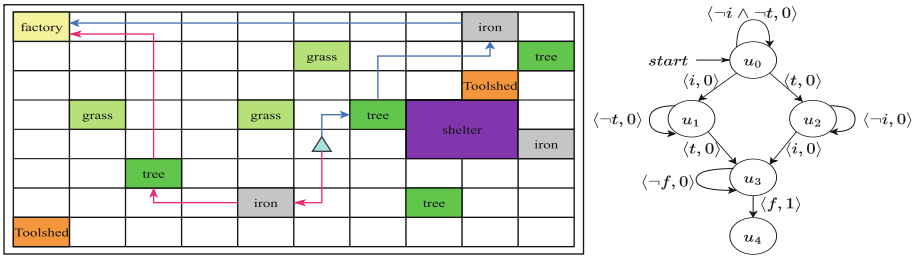
2 HOERM Through Reward Machines

This section will introduce the details of the definition of Reward Machine (RM) and how the proposed HOERM learns strategies through RM.

2.1 Reward Machine

Reward Machine (RM) [6] is a special finite state machine that takes an abstract description of the environment as input and outputs the combination of reward functions under different conditions. The formal definition of RM is below:

Definition 1. (*Reward Machine*): *Given a set of propositional symbols* ($\mathcal{P}$), *a set of environment states* (S), *and a set of feasible actions* (A), *an RM can be defined as a five-tuple:* $RM = \langle U, u_0, F, \delta_u, \delta_r \rangle$, *where* U *signifies a finite set of states,* $u_0 \in U$ *is the initial state, and* F *constitutes a finite set of terminal states such that* $U \cap F = \emptyset$. δ_u *is the state transition function,* $\delta_u : U \times 2^{\mathcal{P}} \to U \cup F$, *and* δ_r *is the reward function outputted by the RM,* $\delta_r : U \to [S \times A \times S \to \mathbb{R}]$.



(a) The Minecraft gridworld

(b) A reward machine for making bridge

Fig. 1. The Minecraft environment and one RM for an example task

Figure 1 depicts a "make-bridge" task and its associated RM in the Minecraft environment. An RM $\mathcal{R}$ starts from an initial state u_0 and is in a certain state u_t ($u_t \in U \cup F$) after the agent moves some steps. Every edge in Fig. 1 (b) is labeled as a tuple $\langle \varphi, r \rangle$, in which φ is a logical formula composed of propositional symbols from $\mathcal{P}$ and r is the reward. When a truth assignment σ satisfies φ (i.e., $\sigma \models \varphi$), $\mathcal{R}$ transitions from state u_t to u_{t+1} and gives a reward of $r = \delta_r(u_t, u_{t+1})$. The σ is a set containing propositions in $\mathcal{P}$, which are true at the state u_t of RM.

To obtain the truth assignment σ, we introduced a labeling function $L : S \times A \times S \to 2^{\mathcal{P}}$. Given a state transition $e = (s, a, s')$, the labeling function $L(s, a, s')$ assigns truth values to the proposition symbols in $\mathcal{P}$. According to the RM definition and $L(s, a, s')$, we can formally describe how to apply RMs in RL.

Definition 2. *MDPRM: A Markov Decision Process with an RM (MDPRM) can be represented by a tuple* $\mathcal{MR} = \langle S, A, p, \gamma, \mathcal{P}, L, U, u_0, F, \delta_u, \delta_r \rangle$*, in which* S*,* A*,* p*, and* γ *are the conventional definitions of an MDP;* $\mathcal{P}$ *is the set of propositional symbols;* $L : S \times A \times S \to 2^{\mathcal{P}}$ *serves as the labeling function; and* U*,* u_0*,* F*,* δ_u*, and* δ_r *are the definitions of the RM.*

In $\mathcal{MR}$, the states of the environment and RM are updated after each step of the agent's execution. After the agent takes an action, then s is transitioned to s', the RM state will also transition to $u' = \delta_u(u, L(s, a, s'))$. Then, the agent receives a reward $r = \delta_r(u, L(s, a, s'))$, which is used to learn and adjust policies.

2.2 Proposed HOERM

This subsection introduces our HOERM for learning MDPRM policies. Our HOERM learns a set of options [5] for each task, focusing on causing RM to transition from one state to another. The high-level policy learns to select the most suitable option, while the low-level policy is modeled using double DQN, responsible for executing actions, and learns options to complete each subtask.

As mentioned above, our HOERM learns an option for each transition $\langle u, u_t \rangle$ between RM states. The initial set of each option is defined as $\mathcal{I}_{\langle u,u_t \rangle} = \{\langle s, u \rangle : s \in S, u \in U\}$. Since option$_{\langle u,u_t \rangle}$ can only be selected to execute when the RM state is u, its policy can be represented as $\pi_{\langle u,u_t \rangle}(a|s)$ by the environmental state s. The termination of an option$_{\langle u,u_t \rangle}$ is defined by Eq. (1), that is, when it transitions to a new RM state or reaches the final state of the task.

$$\beta_{\langle u,u_t \rangle}\left(s', u'\right) = \begin{cases} 1 & \text{if } u' \neq u \text{ or } s' \text{ is terminal} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The goal of option strategy $\pi_{\langle u,u_t \rangle}$ is to guide the agent to complete the option as soon as possible. Thus, we designed an innovative reward mechanism.

$$r_{\langle u,u_t \rangle}\left(s, a, s'\right) = \begin{cases} \delta_r(u)\left(s, a, s'\right) + \gamma\Phi(u_t) - \Phi(u) & \text{if } u_t \neq u \text{ and } u_t = \delta_u\left(u, L\left(s, a, s'\right)\right) \\ \delta_r(u)\left(s, a, s'\right) + r^+ & \text{if } u_t \neq u \text{ and } u_t \neq \delta_u\left(u, L\left(s, a, s'\right)\right) \\ \delta_r(u)\left(s, a, s'\right) + r^- & \text{otherwise} \end{cases} \tag{2}$$

In Eq. (2), $\delta_r(u)\,(s, a, s')$ is the reward function output by RM, which outputs 1 only when it reaches the final state and 0 otherwise. $\Phi(u)$ is the potential function used to measure the proximity of the current state to the terminal state. In simple terms, the closer u is to the terminal state, the larger $\Phi(u)$ is. γ is the discount factor defined in the MDP. The hyper-parameters r^+ and r^- represent the auxiliary reward and punishment, respectively. If the transition (s, a, s') makes the RM state transition from u to u_t $(u_t = \delta_u(u, L(s, a, s')))$, agent will receive a reward of $\delta_r(u)(s, a, s')$, as well as an additional reward of $\gamma\Phi(u_t) - \Phi(u)$, to encourage the agent for completing the RM state transition.

In addition, when the RM state transitions from u to another state $\overline{u}$ $(\overline{u} \notin \{u, u_t\}$, but $\overline{u} \in U)$, the agent receives a reward of $\delta_r(u)(s, a, s')$ and an additional auxiliary reward of r^+. This is because even though the transition is not the intended one, the experience is still valuable for the overall task.

For this reason, the experience replay mechanism includes a unique experience buffer, denoted as $ReplayBuffer_2$. It specifically stores the agent's experiences when triggering high-level events that lead to transition from u to another state $\overline{u} = \delta_u(u, L(s, a, s'))$, where $\overline{u} \notin \{u, u_t\}$ but $\overline{u} \in U$. The two main reasons for this design are as follows: (1) The conventional random sampling from the experience buffer often results in numerous irrelevant experiences, leading to slow training. (2) No matter whether $\overline{u}$ is equal to u_t, the experience gained when the agent triggers a high-level event is valuable, which helps to expedite the learning of policies for completing the subsequent subtask swiftly.

Finally, if the agent fails to trigger any high-level event and the RM state remains unchanged, it will receive an auxiliary punishment (r^-) in addition to the reward $(\delta_r(s, a, s'))$. This negative feedback can motivate the agent to adjust its policy for the subtask as soon as possible.

3 Experiments

In this section, we perform algorithm performance tests on environments with discrete (Minecraft [1]) and continuous (Water World [3]) state spaces. These environments cover both single-task and multi-task scenarios. Three baseline algorithms were used, which are representative, state-of-the-art approaches for combining RMs with reinforcement learning. The first is Q-learning for RMs (QRM), proposed by [4]. The second and third are counterfactual experiences for RMs (CRM) and hierarchical RL for RMs (HRM), proposed by [2].

3.1 Experimental Setup

Each algorithm was run independently three times, and the average performance was reported. The results are presented uniformly in this paper, with four algorithms compared: the red line is the proposed HOERM; the green line is QRM; the sky-blue line is CRM; and the magenta line is HRM. The X-axis corresponds

to the number of training steps, in millions, while the Y-axis represents the average normalized reward across all tasks. Tables 1 and 2 also provide the average and maximum reward values obtained by each algorithm.

3.2 Experiment 1: Results in Minecraft

The first experiment was conducted in the Minecraft environment with a discrete state space, introduced by [1]. As shown in Fig. 1 (a), the environment grid contains the agent and raw materials needed for the tasks, with randomly generated positions. The task set $\Omega = \{\varphi_1, \varphi_2, \ldots, \varphi_9, \varphi_{10}\}$, where $\varphi_i \in \Omega, i \in [1, 10]$ is one task in Minecraft (e.g., the "make-bridge" task).

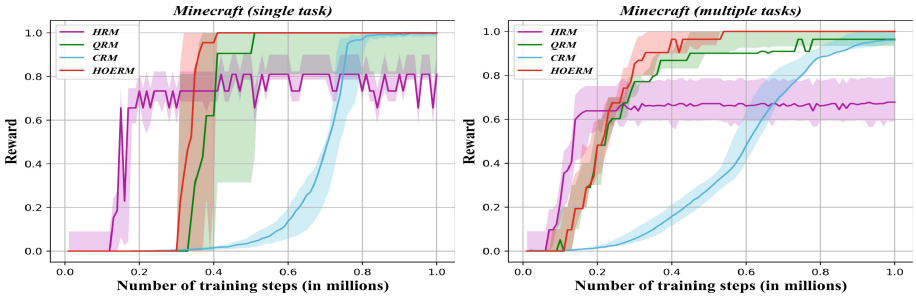


Fig. 2. Results in the Minecraft environment.

Figure 2 and Table 1 show the results obtained by each algorithm in single-task and multitask tests. According to the experimental results, the HRM algorithm has a fast initial learning speed in the two tests. In the single-task test, HRM's average reward is 0.6480, which is only slightly lower than our HOERM's 0.6712. However, HRM's maximum reward is the lowest, only 0.81. The QRM and CRM have a slower speed of obtaining rewards than HRM but result in a maximum reward of 1. The results are similar for the multitask test. On the whole, our HOERM algorithm performs best in both tests. Although the initial learning speed is slightly slower than that of HRM, it obtains the highest values for both average and maximum rewards. The maximum rewards in two types of tests are 1, indicating that HOERM can complete all tasks excellently.

Table 1. Rewards in the Minecraft environment.

Algorithm	single task		multiple tasks	
	maximum reward	average reward	maximum reward	average reward
HOERM	**1.0**	**0.6712**	**1.0**	**0.7746**
QRM	**1.0**	0.6218	0.9642	0.7218
CRM	**1.0**	0.6095	0.9602	0.5803
HRM	0.81	0.6480	0.6779	0.5917

3.3 Experiment 2: Results in Water World

The second experiment was conducted in "Water World", which is an environment with continuous state space and consists of a 2D box containing balls of various colors. We defined 10 tasks in this environment, including touching different specific colored balls. More detailed information can be found in [2].

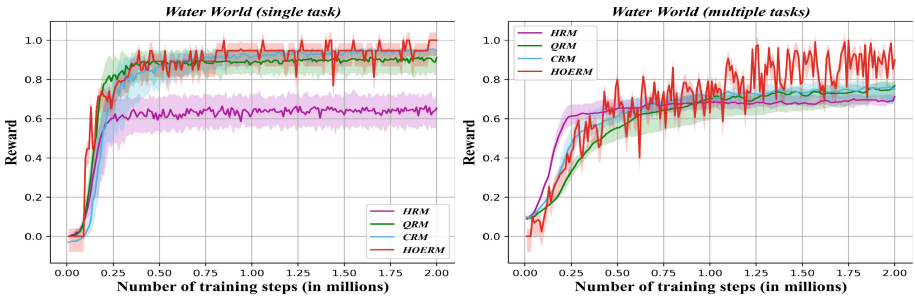


Fig. 3. Results in the Water World environment.

Figure 3 and Table 2 present the result of four algorithms in Water World. In the single-task test, the four algorithms exhibit similar initial learning speeds, but the HRM algorithm still achieves the lowest maximum reward of 0.666. In multitask test, notably, HRM's average reward is 0.6318, which is close to the value for CRM and higher than QRM thanks to its fast initial learning speed. However, HRM's maximum reward of 0.7125 still remains the lowest. Both CRM and QRM do not significantly improve their rewards compared to HRM in this complex environment. Although HOERM has a slightly slower initial learning speed than HRM, the effective sample utilization enhances its task completion capability, resulting in the highest rewards than the three baseline algorithms, demonstrating its excellent performance.

Table 2. Rewards in the Water World environment.

Algorithm	single task		multiple tasks	
	maximum reward	average reward	maximum reward	average reward
HOERM	**1.0**	**0.8571**	**0.9466**	**0.6918**
QRM	0.9137	0.8257	0.7689	0.6038
CRM	0.9553	0.8255	0.7644	0.6391
HRM	0.6660	0.5898	0.7125	0.6318

4 Discussion

In this paper, we introduced the concept of RM and proposed an HRL algorithm. The proposed algorithm effectively leverages RMs and achieves efficient

sample utilization, leading to rapid strategy learning and impressive experimental results. In future work, we will explore the application of RMs in multi-agent problems to tackle more complex tasks effectively by sharing RM knowledge.

Acknowledgements. This study was supported by the National Natural Science Foundation of China (Grant Nos. 62172072).

References

1. Andreas, J., Dan, L., et al.: Modular multitask reinforcement learning with policy sketches. In: 9th International Conference on Machine Learning, Proceedings, pp. 166–175 (2017)
2. Icarte, R.T., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Reward machines: exploiting reward function structure in reinforcement learning. J. Artif. Intell. Res. **73**, 173–208 (2022)
3. Karpathy, A.: REINFORCEjs: WaterWorld demo (2015). http://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html
4. McIlraith, S., Icarte, R.T., Klassen, R.: Using reward machines for high-level task specification and decomposition in reinforcement learning. In: 10th International Conference on Machine Learning, Proceedings, pp. 2107–2116 (2018)
5. Sutton, R.S., et al.: Between MDPs and Semi-MDPs: a framework for temporal abstraction in reinforcement learning. Artif. Intell. **112**(1–2), 181–211 (1999)
6. Toro Icarte, R., Waldie, E., Klassen, T., Valenzano, R., Castro, M., McIlraith, S.: Learning reward machines for partially observable reinforcement learning. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
7. Wiering, M.A., Van Otterlo, M.: Reinforcement learning. Adapt. Learn. Optim. **12**(3), 729 (2012)
8. Zheng, X., Yu, C., Zhang, M.: Lifelong reinforcement learning with temporal logic formulas and reward machines. Knowl.-Based Syst. **257**, 109650 (2022)