Does Hierarchical Reinforcement Learning Outperform Standard Reinforcement Learning in Goal-Oriented Sparse Reward Domains?

Ziyan Luo[*] [†]
ziyan.luo@mail.mcgill.ca
Mila, McGill University

Yijie Zhang[*]
yj.zhang@mail.mcgill.ca
McGill University

Zhaoyue Wang[*]
zhaoyue.wang@mail.mcgill.ca
McGill University

Abstract

Hierarchical Reinforcement Learning (HRL) targets long-horizon decision-making problems by decomposing the task into a hierarchy of subtasks. There is a plethora of HRL works that can do bottom-up temporal abstraction automatically meanwhile learning a hierarchical policy. In this study, we assess the performance of standard RL and HRL within a customizable 2D Minecraft domain with varying difficulty levels. We observed that without prior knowledge, predefined subgoal structures and well-shaped reward structures, HRL methods with automatic option discovery surprisingly do not outperform all standard RL methods in 2D Minecraft domain. We also provide clues to elucidate the underlying reasons for this outcome, e.g., whether HRL methods, incorporating automatic temporal abstraction, can discover bottom-up action abstractions that match the intrinsic top-down task decomposition, often referred to as "goal-directed behavior" in goal-oriented environments[3].

1 Introduction and Background

Reinforcement Learning (RL) is a machine learning paradigm that is widely applied in decision-making problems [25]. However, standard RL faces challenges such as *long horizon* and *sparse reward*, making it difficult to tackle more complex tasks. Benchmarks like the Atari game "Montezuma's Revenge" [12] and Minecraft [1] embody these hurdles, featuring notably sparse and delayed rewards (received only upon goal attainment) while requiring a temporally extended action structure. As an example of such tasks, in Minecraft (shown in Section 2), the agent needs to locate various raw materials on the map, combine them to construct tools, and then effectively utilize these tools to achieve the goal. *Hierarchical Reinforcement Learning* (HRL) tackles such long-horizon decision-making problems by decomposing the task into a hierarchy of subtasks, thus improving learning efficiency [29]. Prior approaches in HRL mainly based on two formalisms [19]: subtasks and options. Subtask-based methods typically decompose the main task into several well-defined subtasks in a top-down manner. These methods often strive to improve policy optimization efficiency with a-prior knowledge of the subtask structure, such as utilizing logic and program [24, 27, 30], reward shaping [8, 16], abstraction from human knowledge [1, 4, 12], and large language models [5]. Contrastingly, "options" [20, 26] represent abstract actions at varying time scales, distinguished by

[*]Equal contribution.
[†]Corresponding author.
[3]Our code is available at: https://github.com/HRL-Mine/HRL-Mine

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

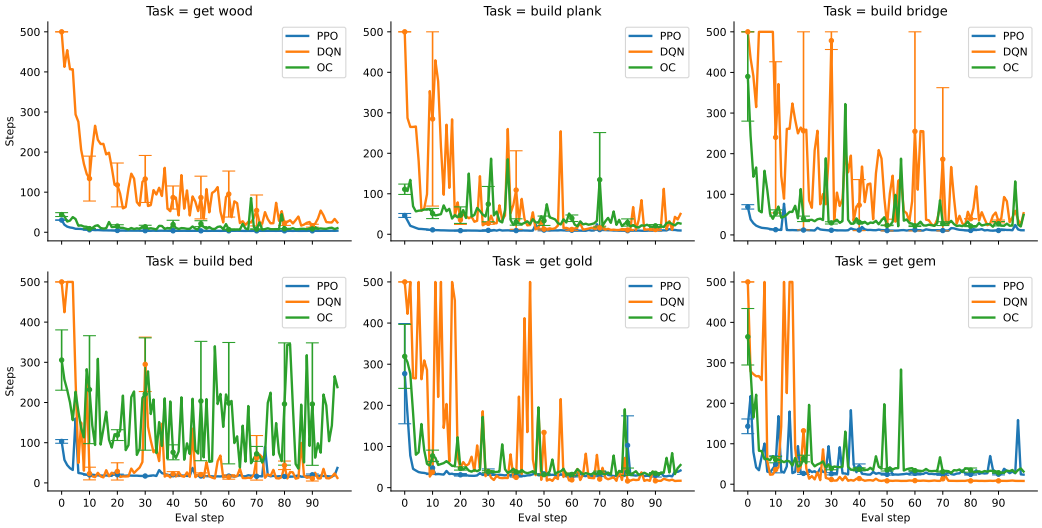

Figure 1: Evaluation on six tasks in Appendix D.2. Each "Eval step" records the *average steps* of 100 evaluation episodes conducted after 100 training episodes. The result is averaged among 3 seeds. Values exceeding 500 are clipped for improved visualization.

their stochastic nature that grants them greater generality. These options are typically introduced in a bottom-up manner without additional knowledge or support.

In this work, we establish a 2D Minecraft environment with explicitly interpretable subgoals (see Appendix D). We investigate standard RL (including Proximal Policy Gradient (PPO, [23]), Deep Q-Learning (DQN, [17]), and Advantage Actor-Critic (A2C, [18])) and HRL methods (see Appendix C) across six tasks (including Option-Critic (OC, [2]), Deep Skill Chaining (DSC, [3, 11]), and Feudal Networks (FuN, [28])). We also provide an empirical analysis to ascertain if and when OC, a representative of option discovery methods, exhibits interpretability and alignment to goal-directed behavior in the Fourrooms domain.

2 Experiment

2.1 Experiment on 2D Minecraft Domain

In our no-frills, easy-to-extend 2D Minecraft environment based on [1], we design several difficulty levels (Appendix D.2) with different numbers of subgoals (e.g. gather or build certain items) that remain undisclosed to the agent: these handcrafted subgoals are not explicitly embedded as pretrained skills or any form of prior knowledge, such as shaped rewards. The implementation detail of our experiment is presented in Appendix C.

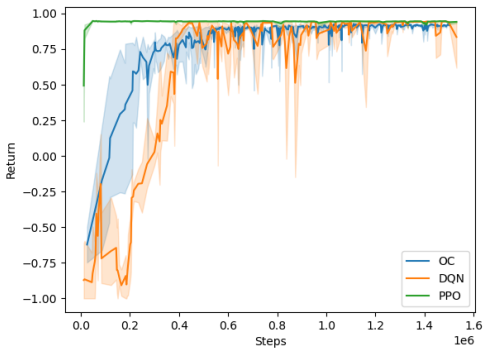

Figure 2: Evaluation on OC with 3 options, DQN, and PPO. Each recorded data point represents the average return of 100 evaluation rollouts. "Steps" represents the training steps when evaluating.

2.2 Experiment on Fourrooms Domain

We revisit the classic navigation problem, Fourrooms, with a slightly different formulation to facilitate the experiment. The implementation detail is presented in Appendix E.

In this section, for convenience, we name the top left room as "a", the top right room as "b", the bottom left room as "c", and the bottom right room as "d". Rooms b and d have direct access to the goal. When in room a, the optimal path is to proceed

to room b through the north doorway and then reach the goal. Conversely, for states in room c, the optimal path involves moving to room d through the south doorway and then reaching the goal.

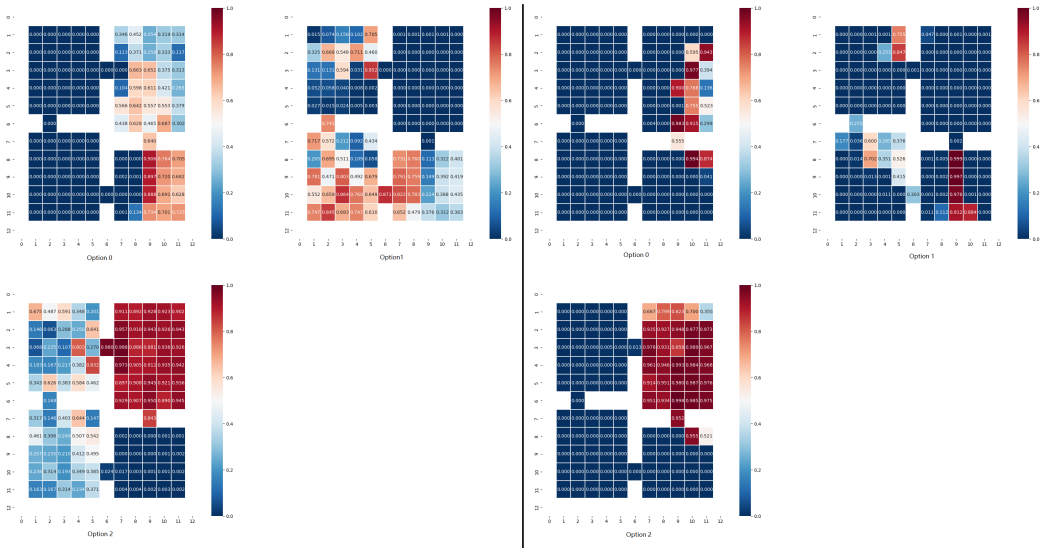

Figure 3: The heat map of termination functions for the 3-option case at around 257K (left) and 1.5M training steps (right). In our visualization, a reddish hue represents a probability closer to 1, while a bluish hue represents a probability closer to 0. Roughly, in the left figure, option 0 corresponds to behavior in rooms a and c (the goal is to exit the rooms), option 1 aligns with behavior in room b, and option 2 corresponds to behavior in room d. In the right figure, however, no such alignment is shown.

In Figure 3 and 4, from the perspective of the termination functions, we can observe some alignment at the early training stage (as shown in Figure 2, DQN and OC do not converge at that time) in some experiments. However, the agent fails to maintain such alignment afterward at 1.5M training steps, though it converges to its best performance then[4]. The same phenomenon is noted in a majority of experiments involving OC with 3 options across various random seeds, although it is not a guaranteed outcome. Figure 2 also shows the superiority of OC over DQN at the initial phase of the training. After approximately 2.5 million time steps, we observe certain undesired behaviors reported by previous work [6], such as frequent switching between options at each time step or consistently utilizing a single option, which sometimes causes a significant drop in performance.

2.3 Main Results

- *PPO handles some HRL tasks surprisingly well.* The satisfying convergence of PPO in all six tasks proves its capability in our sparse-reward HRL tasks. DQN with epsilon-greedy shows more instability than PPO and OC. A2C manages to do well (close to the performance of PPO) in the first three simpler tasks, but often fails (worse than the random agent in Appendix, Table 1) in the more complex tasks. Thus, we exclude the result of A2C in Figure 1 for better interpretability.
- *OC, though designed for HRL tasks, does not perform the best in our sparse reward settings.* The stability in the training of OC is also not optimal among the baselines, likely stemming from the continual, on-the-fly update of termination functions that frequently alters the optimal intra-option policies $\pi^*_{\omega_i}$ (observed from experiment in Section 2.2).
- *OC is not guaranteed to converge to reasonable termination functions*, let alone decomposing the task aligning to human expectation in our sparse reward settings. This may be attributed to OC relying solely on reward-driven temporal abstractions, where *an optimal policy can be realized with any set of options* (See Lemma 1 in Appendix B) in OC. To provide evidence for this divergence, we revisit OC on the deterministic Fourrooms domain (see Appendix E for details). In Section 2.2, the learned termination functions sometimes show alignment with some well-defined behaviors at

[4]Additional results and the comprehensive dynamics of the heat maps and behavior analysis are included in our codebase: `https://github.com/HRL-Mine/HRL-Mine`

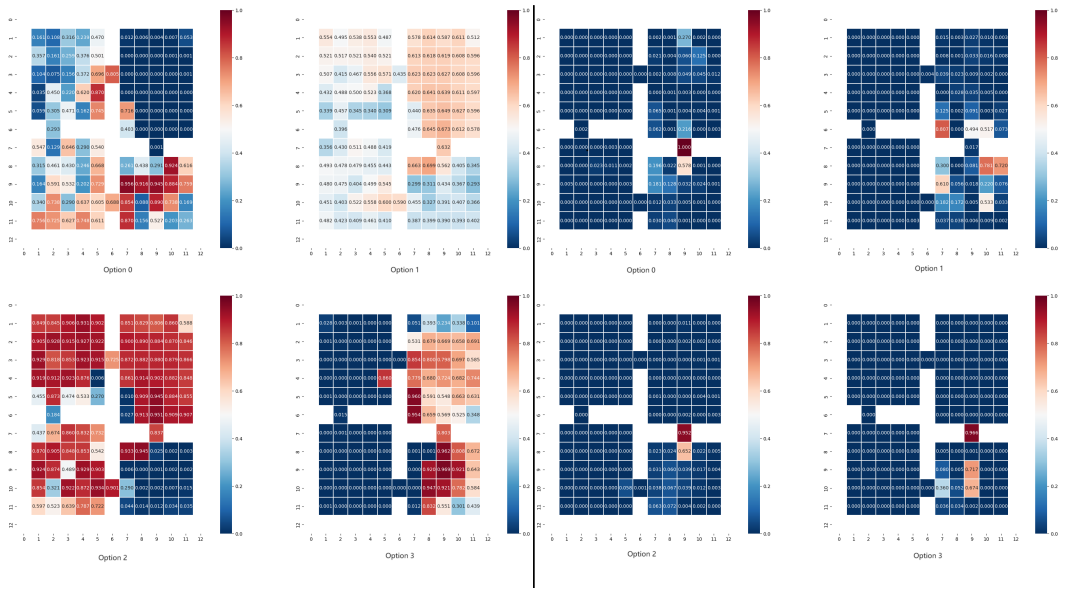

Figure 4: The heat map of termination functions for the 4-option case at around 257K (left four maps) and 1.5M training steps (right four maps). In our visualization, a reddish hue represents a probability closer to 1, while a bluish hue represents a probability closer to 0. Roughly, in the left figure, option 0 corresponds to behavior in room b, option 2 aligns with behavior in room d, and option 3 corresponds to behavior in rooms a and c. Option 1 does not align with any well-defined behavior. In the right figure, however, no such alignment is shown.

the initial stage of training. However, this alignment does not persist throughout the entire training process. We conjecture that: at the onset of training, an accidentally found well-defined set of options facilitates faster learning, surpassing at least the performance of DQN at this time point, as shown in Figure 2. However, as training progresses, reliance on these options is not guaranteed as OC does not inherently provide incentives for improving alignment.

- *Without careful tuning of hyperparameters and design choices, DSC and FuN fail to perform well in our settings.* We exclude them from Figure 1 as they often cannot outperform the random agent.

3 Conclusion and Future Work

Our results underscore that there is significant room for progress in advancing current HRL methods to achieve automatic task decomposition that aligns with human standards in goal-oriented, sparse-reward, and long-horizon domains. A promising avenue for exploration involves formalizing a definition for a "good, better, or best" set of options, emphasizing factors like alignment and adaptability among tasks, thereby guiding algorithms to improve in these directions. Moreover, in sparse-reward settings, efficient temporal credit assignment poses a challenge. Incorporating top-down task decomposition information can prove beneficial in reducing the complexity of the primary problem. Lastly, we intend to bolster the robustness of our findings by conducting training on a larger scale with the extension of "variants" shown in Appendix D.4 and other related work of OC in Appendix F.

References

[1] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pages 166–175. PMLR, 2017.

[2] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[3] A. Bagaria and G. Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2019.

[4] L. C. Cobo, C. L. Isbell, and A. L. Thomaz. Automatic task decomposition and state abstraction from demonstration. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, page 483–490, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0981738117.

[5] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models, 2023.

[6] J. Harb, P.-L. Bacon, M. Klissarov, and D. Precup. When waiting is not an option: Learning options with a deliberation cost. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[7] A. Harutyunyan, W. Dabney, D. Borsa, N. Heess, R. Munos, and D. Precup. The termination critic. *arXiv preprint arXiv:1902.09996*, 2019.

[8] K. Jothimurugan, R. Alur, and O. Bastani. A composable specification language for reinforcement learning tasks. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/f5aa4bd09c07d8b2f65bad6c7cd3358f-Abstract.html`.

[9] K. Khetarpal, M. Klissarov, M. Chevalier-Boisvert, P.-L. Bacon, and D. Precup. Options of interest: Temporal abstraction with interest functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4444–4451, 2020.

[10] M. Klissarov, P.-L. Bacon, J. Harb, and D. Precup. Learnings options end-to-end for continuous action tasks. *arXiv preprint arXiv:1712.00004*, 2017.

[11] G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems*, 22, 2009.

[12] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[14] M. Liu, M. C. Machado, G. Tesauro, and M. Campbell. The eigenoption-critic framework. *arXiv preprint arXiv:1712.04065*, 2017.

[15] M. C. Machado, M. G. Bellemare, and M. Bowling. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*, pages 2295–2304. PMLR, 2017.

[16] O. Marom and B. Rosman. Belief reward shaping in reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2018. URL `https://api.semanticscholar.org/CorpusID:19106988`.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[19] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

[20] D. Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

[21] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

[22] M. Riemer, M. Liu, and G. Tesauro. Learning abstract options. *Advances in neural information processing systems*, 31, 2018.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[24] S.-H. Sun, T.-L. Wu, and J. J. Lim. Program guided agent. In *International Conference on Learning Representations*, 2019.

[25] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[26] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[27] P. Vaezipoor, A. Li, R. T. Icarte, and S. McIlraith. Ltl2action: Generalizing ltl instructions for multi-task rl, 2021.

[28] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017.

[29] Z. Wen, D. Precup, M. Ibrahimi, A. Barreto, B. Van Roy, and S. Singh. On efficiency in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 33:6708–6718, 2020.

[30] Y. Yang, J. P. Inala, O. Bastani, Y. Pu, A. Solar-Lezama, and M. Rinard. Program synthesis guided reinforcement learning for partially observed environments. *Advances in neural information processing systems*, 34:29669–29683, 2021.

Appendix

A Preliminaries

A.1 The Option Framework

The option framework [20, 26] provides important fundamentals in modeling temporal abstraction structures. (Markov) *Options* are temporally extended actions over primitive actions that consist of a triple $\omega = (I_\omega, \pi_\omega, \beta_\omega)$ with I_ω the initiation set of states, $\pi_\omega : S \times A \to [0, 1]$ the option's intra-policy, $\beta_\omega : S \to [0, 1]$ the termination function.

A.2 Option-Critic

Option-Critic (OC, [2]) learns options from scratch by optimizing the *termination functions* (β_ω) as well as the *intra-option policies* (π_ω) in a gradient-based manner, assuming all the options can be applied everywhere. It uses a policy over options (π_Ω) to control which intra-option policy (π_ω) to execute.

- The intra-option policy gradient takes the following form:

$$\frac{\partial L(\theta)}{\partial \theta} = \mathbb{E}\left[\frac{\partial \log \pi_\omega \left(A_t \mid S_t, \Omega_t\right)}{\partial \theta} Q_\omega \left(S_t, \Omega_t, A_t\right)\right] \tag{1}$$

- The termination gradient is as follows:

$$\frac{\partial L(\vartheta)}{\partial \vartheta} = \mathbb{E}\left[-\frac{\partial \beta_\omega \left(S_t, \Omega_t\right)}{\partial \vartheta} \left(A_\Omega \left(S_t, \Omega_t\right)\right)\right] \tag{2}$$

A.3 Deep Skill Chaining (DSC)

Deep Skill Chaining (DSC, [3, 11]) can find options automatically backward from goals. It collects successful trajectories, creates options backward from the goal (or the last option's initial states), and trains the intra-option policy. The initial sets are learned by supervised learning, i.e., doing classification on samples of trajectories. Once the initial set settles, it is considered the first effective option. Then the agent continues to create another option which takes the initial sets of the last option as its goal states. Thus, skills are chained backward until the start state of an agent lies inside the initial sets of an option.

A.4 FeUdal Network (FuN)

FeUdal Network (FuN, [28]) sheds light on defining specific semantics of components of independent MDPs at two levels: manager and worker where the manager is c time steps ahead of the worker. In FuN, the high-level actions (manager) have specific semantics: desired directions in the latent state space, which can be regarded as explicit goals. Such goals are learned in an end-to-end manner. The low-level (worker) reward is composed of extrinsic reward (environmental reward) and intrinsic reward (cosine distance between state variation and the goal).

B Proof sketches

Lemma 1. Given an SMDP $\mathcal{M}$ and its corresponding MDP $\mathcal{M}'$, an optimal policy π^* of $\mathcal{M}'$ can be realized by at least one policy $(\{\pi_{\omega_i}\}, \pi_\Omega)$ in $\mathcal{M}$ with any set of options $\{\omega_i\}$, where $i = 1, 2, \cdots, n$ and $\omega_i = (I_{\omega_i}, \pi_{\omega_i}, \beta_{\omega_i})$.

Proof (sketch). The desired policy $(\{\pi_{\omega_i}\}, \pi_\Omega)$ can be constructed like this: $\forall i$, we let $\pi_{\omega_i} = \pi^*$, i.e., all options have the same intra-option policy π^*. Then, for any n, I_{ω_i}, β_{ω_i}, and π_Ω, the agent follows the same policy as π^*. □

C Implementation and Experiment Details

For the 2D Minecraft domain, we set the timeout for each episode as $t_l = 20,000$ time steps. In each experiment (with a specific random seed), the procedurally generated world map remains consistent, while the initial position of the agent varies across different episodes.

For standard baselines in RL, we adopt PPO [23] and A2C [18] adapted from Stable Baselines3 [21] using the best reported hyperparameters. For Option-Critic, we set the number of options as the number of subgoals of the tasks, while other hyperparameters remain the same as in [2]. A simple three-layered multilayer perceptron (MLP) with a hidden size of 64 is adopted in all compared methods. For DSC, we adopted DQN instead of Deep Deterministic Policy Gradient (DDPG, [13]) to deal with the discrete action space.

For evaluation, as shown in Figure 1, we run each experiment three times with different random seeds and average the result. The agent is evaluated after each 100 training episodes. Each evaluation period lasts for another 100 episodes, and we average the episodic steps taken.

D The 2D Minecraft Domain

D.1 MDP Formulation

The MDP and task formulation of 2D Minecraft are as follows:

Action The primitive action space $\mathcal{A}$ is discrete and can be divided into two parts: 1) *motor actions*, including going up, down, right, and left (if the next state corresponds to a wall, stone, or water, the agent remains in its current location); 2) *crafting action*: including an `USE` action that indicates the agent interacts with the environment, i.e., collecting the resource to an *inventory*, using workshop (to build a tool using collected resources according to a *recipe*) or using tools.

State A state $s \in \mathcal{S}$ is a representation of a snapshot of the world and the inventory. For simplicity, we use a symbolic state representation that encode the world state and the inventory state.

Reward In this environment, the positive sparse reward $R = r_g = 1$ is given only when the goal g is reached, with a small punishment $\epsilon = 1/t_l$ at each time step. That makes the task difficult to be solved with flat, monolithic policies.

Transition Function. To maximize simplicity, we set the MDP deterministic.

Task A *task* τ is specified by the agent's initial state distribution (world initial map, initial position and inventory state) $\rho_\tau : \mathcal{S} \to \mathbb{R}$, the goal g that can be grounded by a task-specific reward function $R_\tau : \mathcal{S} \to \mathbb{R}$. The *difficulty* of a task depends on many factors, e.g., world map size, number of resources, options required, topography, etc.

D.2 Six Tasks, Four Levels

The task specifications are as follows:

Level 1 *Get wood* (1 subgoal): randomly place a few woods; the goal is to get one of them.

Level 2 *Build plank* (2 subgoals): get a wood, and use the wood in the workbench to get the plank; *Build bridge* (3 subgoals): get a wood and an iron, and use the workbench.

Level 3 *Build bed* (4 subgoals): get a wood and a grass, use workbench_0 to convert the wood into a plank, then use workbench_1 to convert the plank and grass into a bed.

Level 4 *Get gold* (5 subgoals): get wood, iron, use the workbench to build a bridge, use the bridge in a right place (if not, the agent need to rebuild), and get gold; *Get gem* (6 subgoals): get wood, iron, use workbench_0 to convert the wood into a stick, then use workbench_1 to convert the stick and iron into an axe, use the axe to break the stone, and get gem.

Among all tasks, the world map size is 10×10, and 4 essential primitive (not synthesized) resources are provided in the map. The initial position of the agent is randomized when the environment resets. Table 1 clearly presents a measurement of the inherent difficulty of each task.

Table 1: Average steps taken among 10,000 episodes of a random agent. No time limit for the random agent is set here. "Fixed" means for each episode, the world map (the placement of resources and workshops) is unchanged, "OOD" means otherwise.

	get wood	build plank	build bridge	build bed	get gold	get gem
Fixed	103.53	478.09	465.27	835.64	1512.32	1686.03
OOD	80.77	438.70	494.23	830.36	1539.17	1721.10

D.3 Example Illustration of 2D Minecraft World

We provide an example illustrating how the task "get gem" is represented in our environment. The boundary of the map is represented as "1". The agent's position is "-1". The resources: iron, wood, and gem are represented as "5", "6", "7" respectively. "4" is the stone that is needed to break using an axe. "2" and "3" represent workshop_0 and workshop_1 respectively.

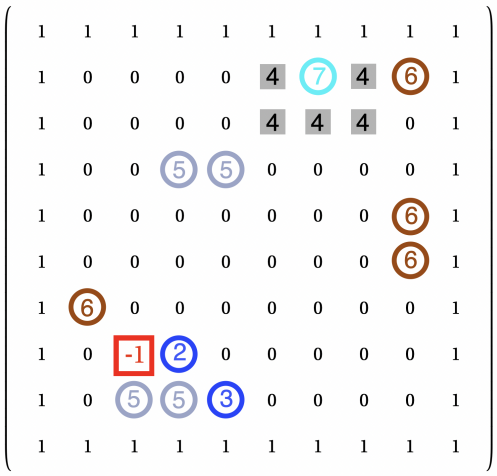

Figure 5: The world map of the task "get gem". In every experiment (with a specific random seed), the positions for "0-7" remain the same, while the initial position of "-1" varies among different episodes.

D.4 Excluded Variants in the Environment

The above-mentioned difficulty levels pertain to "clean" environments, excluding the "variants" as follows:

- Larger or smaller map size than 10×10;
- More or less quantity of each kind of resource;
- Distraction workshops or items that do not contribute to goal achievement;
- Destructive actions that hinders the agent's capability to achieve the goal;
- Out-of-distribution (OOD) environment (e.g., different world maps for different episodes);
- Aleatoric stochasticity.

E The Fourrooms Domain

In this section, we revisit the classic navigation problem, Fourrooms, with a slightly different formulation to facilitate the experiment.

E.1 MDP Formulation

The MDP formulation of this domain is as follows:

Action. The action space $\mathcal{A}$ contains four motor actions including going up, down, right and left.

State. A state $s \in \mathcal{S}$ is a one-hot vector encoding of an index of a possible position on the map, and signals indicating whether subgoals are triggered (only available in multi-task settings to ensure Markov property). Importantly, the map information is not encoded in the state vector. When resetting the environment, the agent is initialized at a random position other than the position of the goal.

Reward. The positive reward 1 is given only when the goal $g \in \mathcal{S}$ is reached. The goal is located at the east doorway. A small punishment ϵ is given at each time step. If a time limit $l = 15\ t^*$ (t^* represents the time steps used following an optimal path) is reached, no reward is given and this episode is truncated.

Transition Function. To maximize simplicity, we set the MDP deterministic.

F Related Works of Option-Critic

In general, OC works on how to automatically learn *a certain number* of options. One main issue of vanilla OC is that, sometimes the learned options switch too often and are collapsed into single actions. A recent extension [6] adds a deliberation cost $c(s, \omega) < 0$ to prevent switching options too often. Another work [7] tries to lower the entropy of termination functions to enhance the *compressibility* of the option's encoding. Proximal Policy Option-Critic (PPOC, [10])is an extension of OC designed for tasks with continuous action spaces, incorporating PPO-styled policy updates. Hierarchical Option-Critic [22] enables a multi-level option structure. Interest Option-Critic (IOC, [9]) further parameterizes the initial condition of options. Eigenoption-Critic [14] ports the idea of eigenoption [15] to OC framework.