Learning Higher Order Skills that Efficiently Compose

Anthony Z. Liu 1 2  Dong-Ki Kim 2  Sungryull Sohn 2  Honglak Lee 1 2

Abstract
Hierarchical reinforcement learning allows an agent to effectively solve complex tasks by leveraging the compositional structures of tasks and executing a sequence of skills. However, our examination shows that prior work focuses on learning individual skills without considering how to efficiently combine them, which can lead to sub-optimal performance. To address this problem, we propose a novel framework, called second-order skills (SOS), for learning skills to facilitate the efficient execution of skills in sequence. Specifically, second order skills (which can be generalized to higher orders) aim to learn skills from an extended perspective that takes into account the next skill required to accomplish a task. We theoretically demonstrate that our method guarantees more efficient performance in the downstream task compared to previous approaches that do not consider second-order skills. Also, our empirical experiments show that learning second-order skills results in improved learning performance compared to state-of-the-art in baselines across diverse benchmark domains.

1. Introduction
Real world compositional tasks require long-horizon planning and can be difficult for reinforcement learning (RL) approaches to learn successful policies to solve them. An effective approach for solving these long-horizon tasks is hierarchical reinforcement learning (HRL) (Sutton et al., 1999), which decouples the complex learning problem into multiple simpler subproblems. In HRL, low level policies (also referred to as skills) are learned to solve these subproblems while a high level policy composes the learned skills by selecting different skills based on the state to solve the downstream task. Prior work in HRL has studied the problem of learning skills and composing the learned skills independently (Oh et al., 2017; Andreas et al., 2016; Ahn et al., 2022). These skills are often learned in an unsupervised setting, where samples are easy to gather, and then fine-tuned in a downstream task (Eysenbach et al., 2018; Gregor et al., 2016; Choi et al., 2021).

However, we highlight in this paper that when learning these skills in an unsupervised setting, it can be easy to learn skills that do not compose together optimally for the downstream task. This sub-optimality arises whenever there is ambiguity in how the agent should execute the skill, and when the agent only optimizes to maximize the individual skill’s reward. As a result, only maximizing individual skill reward can be detrimental when the agent must execute other skills in the future for the downstream task. Figure 1 shows an example of this sub-optimality issue, where the agent must first gather wood and then use a workbench to complete the task of making planks. Note that the example has ambiguity in which wood the agent should gather (i.e., either the left or right wood). If the agent looks ahead and considers the second step to complete the task, the agent can infer that gathering the wood closer to the workbench is a faster path than gathering the other wood.

With this insight, this paper introduces learning of a new
Learning Higher Order Skills that Efficiently Compose

form of skills that maximize individual skill reward while also maximizing future rewards for other skills. We call these dual-maximizing future rewards skills second-order skills (SOS). Formally, every second-order skill is made of a target skill and a future skill, where the agent executes the target skill while maximizing future rewards for the future skill. This enables the agent to “look-ahead” to the future skill, enabling the agent to execute the target and future skill for a higher return than executing first-order skills for downstream tasks. Further, by iteratively applying second-order skills, we can extend the idea of second-order skills to downstream tasks. Analogous to how second-order skills are more efficient than first-order skills, composing nth order skills for downstream tasks is more efficient than composing n − 1th order skills.

In summary, we make the following primary contributions:

- **Novel Skill Learning.** We define second-order skills (Section 3.1) and introduce the SOS algorithm for learning them using any off-policy RL algorithm (Section 3.2).

- **Theoretical Analysis.** We theoretically present that high level policies that execute second-order skills will be more efficient than only executing first order skills.

- **Empirical Evaluation.** We empirically show the effectiveness of learning second-order skills using two benchmark domains: the crafting domain (Andreas et al., 2016) and minigrid placement (Sohn et al., 2018; Liu et al., 2022).

2. Problem Statement

2.1. Hierarchical Reinforcement Learning

Our HRL setting considers a finite episodic MDP \((S, A, P, R, \gamma, T)\), where \(S\) is the set of states, \(A\) is the set of actions, \(P\) is the state transition probability, \(R\) is the reward function, \(\gamma\) is the discount factor, and \(T\) is the episodic horizon (Sutton and Barto, 1998; Sutton et al., 1999). A hierarchical agent aims to maximize the expected return for the downstream task by splitting the problem of learning into two: (1) Instead of learning behavior for the entire MDP, the agent learns a set of sub-behaviors \(G\) called skills. Specifically, the agent learns a low-level policy \(\pi_{LL}(a|s, g)\) for each skill \(g \in G\), where \(a \in A\) and \(s \in S\), by maximizing the skill rewards \(r_g(s, a)\). Each low-level policy also learns a termination policy \(\beta_{LL}(\text{done}|s, a, g)\), predicting whether the current skill \(g\) is done (0 or 1) given the current state \(s\) and action \(a\). (2) The HRL agent also learns a high-level policy \(\pi_{HL}(g|s)\), which selects the best low-level policy (i.e., skill) to maximize the task return. The high-level policy executes the selected low-low policy \(g\) until it terminates.

2.2. Skill Pre-Training

We consider a variant of the HRL problem, where the agent learns the low-level skills \(G\) in isolation to the downstream task \(R\). This setting is considered in a variety of related work regarding unsupervised skill discovery (Eysenbach et al., 2018; Gregor et al., 2016) and goal conditioned RL (Nachum et al., 2018). These methods learn skills by training an RL policy on a given or inferred reward and use the learned skills to enable sample efficient learning for HRL. Following prior work (Oh et al., 2017; Andreas et al., 2016; Ahn et al., 2022), we make two minor assumptions to make learning and deriving theoretical properties easier: (1) that skill rewards have already been inferred, and (2) that these skill rewards are non-negative (see Appendix A.2 for more details).

Ambiguity Issue. A critical issue in the unsupervised learning setting is the ambiguity problem, which occurs whenever a skill has multiple paths to successful execution. Specifically, when the agent executes a skill that has multiple paths of execution, the agent may arrive in a state that is sub-optimal for the downstream task. For example, Figure 1 shows the ambiguity in choosing which wood to collect, where getting left or right wood successfully terminates the skill but either choice results in a different performance. We address this ambiguity issue by introducing the concept of second-order skills in the next section.

3. Approach

This section formally defines second-order skills (SOS) and introduces a new algorithm for learning them using any off-policy RL algorithm. Intuitively, SOS learns a second-order skill \(g_1 \rightarrow g_2\) that executes a skill \(g_1\) while maximizing future return for a next skill \(g_2\). We also present theoretical results, showing that when using second-order skills for HRL, SOS allows the agent to achieve a higher return than first-order skills on downstream tasks. Finally, we generalize our second-order approach to nth-order skills that can be defined by recursively applying second-orders to other second-order skills. Similarly, these nth-order skills can be learned using our SOS algorithm and can achieve higher theoretical HRL performance.

3.1. Higher-Order Skills and HRL

Second-Order Skills. Formally, given two skills \(g_1\) and \(g_2\), we define the second-order skill \(g_1 \rightarrow g_2\) to be the skill that trains under the same MDP as skill \(g_1\) but instead maximizes the following second-order reward function:

\[
r_{g_1 \rightarrow g_2}(s, a) = \mathbb{E}_{s' \sim P(s, a)} \left[ \begin{cases} \hat{Q}_{g_2}(s, a) & \text{if } s' \in S_{g_1} \\ 0 & \text{else} \end{cases} \right]
\]
where $S_{g_1}$ denotes a latent set of goal states determining when the agent has terminated the skill $g_1$ successfully, and $Q_{g_2}(\cdot, \cdot)$ denotes the estimated Q value for the skill $g_2$. Intuitively, Equation 1 states that when the agent executes $g_1$ successfully, it receives a reward of $Q_{g_2}(s, a)$, enabling the second-order skill $g_1 \rightarrow g_2$ to look-ahead toward $g_2$ while executing $g_1$. We show a visualization of value functions resulting from second-order skills reward in Section 4.4.

The notion of a successful skill is important when composing skills together for downstream tasks. The success function is generally unknown to the agent and must be estimated. In practice, we can estimate $S_{g_1}$ using the reward $r_{g_1}$ as the two are closely related:

$$S_{g_1} = \{ s \in S \mid r_{g_1}(s, a) > \rho \},$$

where $\rho$ is a hyperparameter.

**nth-Order Skills.** Because $g_1 \rightarrow g_2$ is itself a skill, nth-order skills can be defined by recursively applying second-order skills. For example, a third-order skill $g_1 \rightarrow g_2 \rightarrow g_3$ can be learned in the same way as second-order skills by first learning $g_2 \rightarrow g_3$ and then learning $g_1 \rightarrow (g_2 \rightarrow g_3)$.

**nth-Order HRL.** Similar to HRL, nth-order HRL learns both a low-level and high-level policy but the low-level policy learns nth-order skills. Specifically, we learn (1) A low-level policy for each nth order skill: $\pi_{nL}(a|s, g_1 \rightarrow g_2 \rightarrow \ldots \rightarrow g_n)$. The termination policy for the nth order skill is the same as the target skill’s termination skill: $\beta_{nL}(\text{done}|s, g_1)$, (2) Then, the high-level policy is learned as $\pi_{nH}(g_1 \rightarrow g_2 \rightarrow \ldots \rightarrow g_n|s)$, which selects the nth-order skills for the low-level policy.

### 3.2. Learning Higher-Order Skills

This section details how second-order skills can be learned based on an off-policy RL algorithm. As with the definitions in Section 3.1, nth-order skills can be learned by recursively applying our learning algorithm on second-order skills.

**Universal Value Function Approximation.** Following (Schaul et al., 2015a), we use a single neural network to approximate value functions for each first-order and second-order MDPs: $\psi : S \times A \times G \times G \rightarrow \mathbb{R}$, which takes state, action, and embeddings for two skills (sub-skill and final skill). Note that if the sub-skill and final skill are equal, $\psi$ outputs the values that correspond to the first-order skill, so our definition of second-order skills generalizes to first-order skills.

**Off-Policy RL with Second-Order Skills.** We give the pseudo-code for learning second-order skills using any off-policy RL algorithm (e.g., double DQN (Hasselt et al., 2015)) in Algorithm 1 in the appendix. Using the current Q estimate $\psi$, we estimate the second-order reward using the current Q estimate $\psi$. 

### 3.3. Theoretical Properties

**Optimality.** We present the main theoretical result in Theorem 3.1, showing that second-order skills are more efficiently executed than first-order skills.

**Theorem 3.1.** Let $g_1$ and $g_2$ be two skills with non-negative rewards $r_{g_1}$ and $r_{g_2}$. Let $g_1 \rightarrow g_2$ be the second order skill for $g_1$ and $g_2$. Suppose there is a downstream task with a reward function $R$ that requires skills $g_1$ and $g_2$ to be executed in sequence. Let $\pi_{2}^{*}$ be the optimal policy for each skill, and trajectories $\tau_1$ and $\tau_2$ are generated by executing skills in sequence by using first-order and second-order skills, respectively:

$$\tau_1 \sim \{(s_1, \pi_{1}^{*}(s_1)), \ldots, (s_i, \pi_{1}^{*}(s_i)), (s_{i+1}, \pi_{2}^{*}(s_{i+1})), \ldots\}$$

$$\tau_2 \sim \{(s_1, \pi_{1}^{g_1 \rightarrow g_2}(s_1)), \ldots, (s_j, \pi_{g_1 \rightarrow g_2}(s_j)), (s_{j+1}, \pi_{g_2}(s_{j+1})), \ldots\}$$

Then the return of second-order skill trajectory $\tau_2$ is always more optimal or equal to the return of first-order skill trajectory $\tau_1$:

$$R(\tau_2) \geq R(\tau_1),$$

where $R$ refers to the expected return of a trajectory.

Our proof is shown in Appendix A.3.

**Corollary 3.2.** Any downstream task that requires skills to be executed in some order can be more efficiently solved using second-order skills than first-order skills. Similarly, nth-order skills can solve these downstream tasks more efficiently than n′th-order skills for any $n' < n$.

This can be shown by recognizing that any order of skill execution on a downstream task can be made more efficient by using the result in Theorem 3.1.

### 4. Experiments

**4.1. Environments**

**Minigrid Placement.** Inspired by block placement games, such as Sokoban (Kartal et al., 2021) and PushWorld (Kansky et al., 2023), we create a simple placement environment in the popular Minigrid Gymnasium environment (Chevalier-Boisvert et al., 2018) (see Figure 2). In this environment, an agent must place two balls (red and blue) into a goal area. However, this task is not trivial to solve as the agent must think ahead and put the first ball further into the goal space so that the agent is not blocked when placing the next ball into the remaining goal space. Two skills are given to the agent: placing the red/blue balls into the goal area. We expect the first-order skills to be short-sighted and immediately place the balls in the closest position, rendering
Learning Higher Order Skills that Efficiently Compose

Figure 2. Visualizations of the Minigrid Placement and Crafting domain (e.g., make plank task) in this paper. A sequence of skills is required to complete a goal in each domain.

the HRL agent unable to solve this environment (since the agent can no longer place the next ball).

Crafting. We implement the Crafting domain from Andreas et al. (2016) in the Minigrid Gymnasium environment. This domain is inspired by minecraft, where the agent must craft various objects using various materials and tools in the environment (see Figure 2). We give the agent 6 low-level skills to learn: get wood, get grass, get iron, use toolshelf, use workbench, use factory. Using these skills, the agent must then learn how to execute the following 8 downstream tasks (we use the make tasks from Andreas et al. (2016)): make plank, make stick, make cloth, make rope, make bridge, make bed, make axe, make shears. The agent is also given a sketch (or a plan) for which low-level skills to execute in order to solve each downstream task. For example, to solve make stick, the agent is given the sketch: [get wood, use workbench]. The full sketch details can be found in the appendix (Andreas et al., 2016).

4.2. Implementation

We learn each environment using a double DQN (Hasselt et al., 2015) with prioritized experience replay (Schaul et al., 2015b), and dueling networks (Wang et al., 2015). We use a similar neural network in each environment: the network uses a 3-layer MLP over a concatenated representation of the state, and one hot encoding of skill(s) (one skill for first-order, and two skills for second-order). We give more details on the architecture in Appendix A.5.

4.3. Results

We trained skills on each environment (Minigrid Placement and Crafting) for 5 million and 10 million timesteps respectively. For the baseline, all timesteps were used to train first-order skills. For SOS, timesteps were used to train first-order and second-order skills in parallel. Then, we trained a high-level DQN policy which executes the pretrained skills on the downstream task. We show the results of the high-level policy in Table 1 and learning curve in Figure 3. The full per-task return for Crafting is shown in the Appendix, Figure 5.

On Placement, we found that first-order skills were unable to solve the downstream task at all (total return of 0) while second-order skills solved the task (total return of 1). This is because the first-order skills always greedily placed the balls in the nearest goal position, which blocks the agent from placing the next ball. The second-order skills allowed the agent to “look-ahead” and place the first ball in a further position, so that the next ball could be placed correctly.

On Crafting, we also found that first-order skills performed worse than second-order (0.67 ± 0.04 to 0.82 ± 0.03 return over 8 seeds). Similar to what we showed in the Crafting example, Figure 1, we found that second-order skills could find more optimal paths to complete the tasks. We also found that second-order skills enabled the agents to solve tasks at a higher rate (refer to Figure 5 in the Appendix), where the second-order skills can consistently solve the difficult long-horizon tasks better (make bridge, make bed, make axe, and make shears). We believe this is due to another property of second-order skills: executing second-order skills leaves agents in a state that it is unlikely to get stuck on, as they maximize the future return on other skills. This enables the high-level agent to succeed since the second-order skills are more likely to succeed.

4.4. Analysis

Second-Order Skills Visualization. We visualize the optimal value functions of second-order skills in Figure 4 in an imagined environment similar to Crafting, where an agent must travel to subgoals α then β. We can see from Figure 4 that \( V^*_{α→β} \) can be thought of as a “reweighted” version of \( V^*_α \), but with the values from \( V^*_β \). From this Figure, we can also see that when an agent must visit α then β, the agent should (most of the time) visit α2 rather than α1, since it is closer to β and the value of \( V^*_{α→β} \) is higher for α2.

Oracle nth-Order Implementation. We also implemented oracle nth skills for skills orders 1 through 4. We evaluated
Learning Higher Order Skills that Efficiently Compose

Figure 3. Episode returns of the HRL agent after training the low level skills for 5M and 10M environment steps for Placement and Crafting respectively. We plotted the mean and standard deviations over 8 seeds. For Crafting, we plot the mean return over all 8 Crafting tasks.

Figure 4. A visualization of a toy environment similar to Crafting where the agent must visit $\alpha$ and $\beta$. We visualize the optimal value functions for the skills $\alpha$, $\beta$ and second-order skill $\alpha \rightarrow \beta$. The values for second-order skill $\alpha \rightarrow \beta$, $V^*_\alpha \rightarrow \beta$ indicate that the agent should visit $\alpha_2$ most of the time if the agent will visit $\beta$ afterward.

Table 2. nth-Order HRL Theoretical Return

<table>
<thead>
<tr>
<th></th>
<th>Placement</th>
<th>Crafting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st-order</td>
<td>0</td>
<td>0.89 ± 0.00</td>
</tr>
<tr>
<td>2nd-order</td>
<td>1</td>
<td>1.01 ± 0.00</td>
</tr>
<tr>
<td>3rd-order</td>
<td>1</td>
<td>1.02 ± 0.00</td>
</tr>
<tr>
<td>4th-order</td>
<td>1</td>
<td>1.02 ± 0.00</td>
</tr>
</tbody>
</table>

We find that in Placement, since there are only 2 skills needed, 2nd-order skills can immediately solve the task. However, in Crafting, the maximum number skills needed to solve a single downstream task is 4. Even so, there are diminishing returns from 2nd-order to 3rd-order to skills. 3rd and 4th order skills are indistinguishable. This implies that for many tasks, second-order skills gives us sufficient approximations of policies for $n$th-order skills.

5. Related Work

Unsupervised Skill Discovery. Unsupervised skill discovery aims to learn skills without the necessity of manually defining and tuning reward functions for desired behaviors. One of the most common approaches to the unsupervised skill discovery problem is to maximize the mutual information (MI) between skill latent variables and states (Gregor et al., 2016; Achiam et al., 2018; Eysenbach et al., 2018; Hansen et al., 2019; Sharma et al., 2019; Choi et al., 2021; Zhang et al., 2021), leading to the discovery of diverse and distinguishable behaviors. The works has been extended to use different posterior models (Hansen et al., 2019), spectral normalization (Choi et al., 2021), and variants of the MI identity (Zhang et al., 2021) to further improve the skill performance. However, these methods share a limitation: they do not consider the behavioral optimality when the learned skills are combined for down-stream tasks. Our work propose a novel framework designed to learn skills that can be executed efficiently when combined together.

Option-Based HRL. The option framework studies discovering temporally extended high-level actions, called options,
to achieve efficient learning (Sutton et al., 1999; Precup and Sutton, 2000). By reducing the effective number of decision makings, options improve an agent’s learning efficiency for solving a long-horizon task (Omidshafiei et al., 2018). Recent frameworks show that an agent can learn both option and termination policies end-to-end without requiring prior knowledge (Bacon et al., 2017; Riemer et al., 2018; Abdulhai et al., 2022). Our main idea of extending the perspective of skills by considering the next skill also applies to option learning settings.

Goal-Conditioned HRL. Another related work is the goal-conditioned setting, where a high-level policy generates a sequence of subgoals for a low-level policy to follow (Kulkarni et al., 2016). Nachum et al. (2018) focused on improving learning efficiency using off-policy correction or hindsight mechanisms. Chane-Sane et al. (2021); Lo et al. (2022); Zhang et al. (2020) proposed efficient subgoal sampling methods to avoid challenges of sparse reward when the subgoal is hard or impossible to reach. Savinov et al. (2018); Laskin et al. (2020); Huang et al. (2019); Hoang et al. (2021) proposed to model the relationship between subgoals in a graph form to enable navigating between subgoals via planning. However, these approaches learn goal-conditioned policy by optimizing to reach any state that belongs to the given subgoal, which may induce a sub-optimal behavior (Nachum et al., 2019). Thus, our work propose a novel way to improve skill learning to enable more efficient composition of skills in down-stream tasks.

6. Conclusion

In this paper, we have introduced second-order skills (SOS), which learns second-order skills to efficiently compose for solving downstream tasks. Our key idea is to extend the perspective skills by looking ahead to next skills that will be executed after the current. We evaluated our method of learning second-order skills on the gridworld benchmark domains and showed that SOS performs more effectively than the first-order baseline.

References


Learning Higher Order Skills that Efficiently Compose


A. Appendix

A.1. Second-Order Skills Pseudocode

Algorithm 1 Learning Second-Order Skills Pseudocode

1: given
2: A distribution of skills or goals $G$.
3: An off-policy RL algorithm $A$ $\triangleright$ DQN, DDPG, etc.
4: with Q network $\psi : S \times A \times G \times G \rightarrow \mathbb{R}$. $\triangleright \psi$ takes a state, action, goal, and final goal
5: A reward function $r : S \times A \times G \rightarrow \mathbb{R}$.
6: A skill success threshold $\rho \geq 0$.
7: Initialize $A$ and Q network $\psi$.
8: Initialize replay buffer $D$.
9: for episode $= 1, \ldots, M$ do
10: Sample a goal $g$ and final goal $g'$ from $G$.
11: for $t = 1, \ldots, T$ do
12: Sample and execute action $a_t$ using policy $A$:
13: $a_t \leftarrow \pi(s_t \| g \| g')$. $\triangleright \|$ denotes concatenation
14: Let $r_t \leftarrow r(s_t, a_t, g)$.
15: Let $t$ denote whether the transition is terminal.
16: Store the transition $(s_t \| g \| g', a_t, r_t, s_{t+1} \| g \| g', t)$ into $D$.
17: if update then
18: Sample a mini-batch $B$ from replay buffer $D$. $\triangleright$ Convert $B$ to a second order mini-batch
19: Let $B_{SOS} = \{\}$. $\triangleright$ If second order
20: for $(s \| g \| g', a, r, s' \| g \| g', t)$ in $B$ do
21: if $g \neq g'$ then
22: $r \leftarrow I[r > \rho] \psi(s \| g \| g', a)$.
23: $t \leftarrow t \lor (r > \rho)$.
24: Append $(s \| g \| g', a, r, s' \| g \| g', t)$ to $B_{SOS}$.
25: Perform an optimization step on $\psi$ with $B_{SOS}$ using $A$.

A.2. Discussion of Second-Order Reward

Non-Negativity. Following prior work (Oh et al., 2017; Andreas et al., 2016; Ahn et al., 2022), we assume that skill rewards are non-negative. Sparse or goal-reaching reward, commonly used in prior work and are used here, are already non-negative, as they give positive rewards when a goal or sub-goal is reached, and zero otherwise.

We also note that many of skill rewards (Eysenbach et al., 2018; Achiam et al., 2018) can also be translated into a non-negative reward using a monotonic function such as $\tan$, or simply flooring the reward such as $r' = \max(0, r)$. However, as this non-trivially changes the skills learned, we leave investigating how to translate reward for second-order skills to future work.

The non-negativity assumption is required since second-order skills may not be beneficial in specific cases when rewards are negative. Specifically, consider two skills $g_1$ and $g_2$, and a second-order skill $g_1 \rightarrow g_2$. Then, under Equation (1), $g_1 \rightarrow g_2$ could learn behavior that accumulates negative reward for $g_2$ before $g_1$ is completed.

$\rho$-Threshold Value. Recall that we estimate the success of a skill using the approximation in Equation (2). The choice of the threshold $\rho$ is set as a hyperparameter. Note that in sparse reward settings, $\rho = 0$ is sufficient, as the reward $r$ is greater than zero only when success is achieved. $\rho$ can also be learned by taking setting $\rho$ to be the $X$th percentile reward observed by the agent.

In the most general scenario, success can be modeled using some learned distribution:

$$S_{g_1} \sim p_\theta(\cdot \mid s, a)$$  (6)

We leave generalizations of success and second-order skills to future work.
A.3. Proof of Theorem 3.1

Before we give the proof of Theorem 3.1, we state our formal definition of a reward function that requires skills to be executed in sequence.

**Definition A.1.** A reward function $R$ is a reward function that requires skills $g_1$ and $g_2$ to be executed in sequence if it follows two properties:

1. $R$ only gives reward if $g_2$ is completed (i.e. $s \in S_{g_2}$) after $g_1$ is completed (i.e. $s \in S_{g_1}$). (With the implicit assumption that history of completion is tracked in the MDP).

2. $R$ is “aligned” with $g_2$. We specify that behavior for maximizing the the skill reward for $g_2$, $r_{g_2}$, also maximizes $R$ when $g_1$ is completed. Formally: Let $\pi_{g_2}$ be a policy for $g_2$. For some state $s$, if $g_1$ is completed ($s \in S_{g_1}$), and $\tau_\pi$ is the trajectory generated by $\pi$ on $s$, then

$$R_{r_{g_2}}(\tau_\pi) > R_{r_{g_2}}(\tau_{\pi'}) \implies R_R(\tau_\pi) > R_R(\tau_{\pi'})$$  \hspace{1cm} (7)

where $R_x$ is the expected return for task $x$.

Following this definition, we give the proof for Theorem 3.1 below:

**Proof.** (For ease of reading, we will restate the conditions.) We have skills $g_1$ and $g_2$ with non-negative rewards $r_{g_1}$ and $r_{g_2}$. We have a downstream task $R$ which requires $g_1$ and $g_2$ to be executed in sequence (defined above).

Let $\pi^*_g$ be the optimal policy for a skill $x$, (including second-order $g_1 \rightarrow g_2$). For notation, we denote $\tau_1(s)$ as the first-order trajectory generated by first completing $g_1$ using $\pi^*_g$, then completing $g_2$ using $\pi^*_g$, and $\tau_2(s)$ as the second-order trajectory generated by first completing $g_1$ using $\pi^*_g$, then completing $g_2$ using $\pi^*_g$. I.e.

$$\tau_1(s_1) \sim \{(s_1, \pi^*_g(s_1)), \ldots, (s_i, \pi^*_g(s_i)), (s_{i+1}, \pi^*_g(s_{i+1})), \ldots\}$$

and

$$\tau_2(s_1) \sim \{(s_1, \pi^*_g \rightarrow g_2(s_1)), \ldots, (s_j, \pi^*_g \rightarrow g_2(s_j)), (s_{j+1}, \pi^*_g(s_{j+1})), \ldots\}$$  \hspace{1cm} (8)

We will show that for all $s \in S$, $R_R(\tau_2(s)) \geq R_R(\tau_1(s))$. We break this into two cases:

1. If $g_1$ is already completed: $s \in S_{g_1}$. Then the two trajectories are equal, since they both execute $\pi^*_g$, so $R_R(\tau_2(s)) = R_R(\tau_1(s))$.

2. If $g_2$ is not completed: $s \notin S_{g_1}$.

Let $s'_1$ denote the state reached after executing $\pi^*_g$ on $s$, and let $s'_2$ denote the state reached after executing $\pi^*_g \rightarrow g_2$ on $s$. Note that $s'_1 \in S_{g_1}$ and $s'_2 \in S_{g_2}$.

Then, we can show that $R_R(\tau_2(s'_2)) \geq R_R(\tau_1(s'_1))$. We show by the contrapositive. Suppose that there is some $s'_1$ and $s'_2$ such that $R_R(\tau_2(s'_2)) < R_R(\tau_1(s'_1))$.

Then by the contrapositive on the Definition in Equation (7), $R_{r_{g_2}}(\tau_2(s'_2)) < R_{r_{g_2}}(\tau_1(s'_1))$. However, this contradicts with our definition of the second-order skill $g_1 \rightarrow g_2$, and the optimal policy for the skill $\pi^*_g \rightarrow g_2$. Since, the path to $s'$ is strictly higher rewarding than $s'_2$ according to the second-order reward definition in Equation (1).

Thus, $R_R(\tau_2(s'_2)) \geq R_R(\tau_1(s'_1))$, and it directly follows that $R_R(\tau_2(s)) \geq R_R(\tau_1(s))$, \hfill $\square$
A.4. Hyperparameters

<table>
<thead>
<tr>
<th>Minigrid Placement Hyperparameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>discount $\gamma$</td>
</tr>
<tr>
<td>learn start</td>
</tr>
<tr>
<td>epsilon greedy</td>
</tr>
<tr>
<td>learning rate</td>
</tr>
<tr>
<td>gradient norm clip</td>
</tr>
<tr>
<td>replay frequency</td>
</tr>
<tr>
<td>target update frequency</td>
</tr>
<tr>
<td>batch size</td>
</tr>
<tr>
<td>prioritized replay weight</td>
</tr>
<tr>
<td>prioritized replay exponent</td>
</tr>
<tr>
<td>max memory buffer size</td>
</tr>
<tr>
<td>success threshold $\rho$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Crafting Hyperparameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>discount $\gamma$</td>
</tr>
<tr>
<td>learn start</td>
</tr>
<tr>
<td>epsilon greedy</td>
</tr>
<tr>
<td>learning rate</td>
</tr>
<tr>
<td>gradient norm clip</td>
</tr>
<tr>
<td>replay frequency</td>
</tr>
<tr>
<td>target update frequency</td>
</tr>
<tr>
<td>batch size</td>
</tr>
<tr>
<td>prioritized replay weight</td>
</tr>
<tr>
<td>prioritized replay exponent</td>
</tr>
<tr>
<td>max memory buffer size</td>
</tr>
<tr>
<td>success threshold $\rho$</td>
</tr>
</tbody>
</table>

A.5. Network Architecture

For the Crafting and Placement environments, we use a similar architecture: Each position in the grid is encoded into an object ID and mapped into a learned encoding. The grid encoding is flattened, then concatenated with a one-hot encoding of the direction of the agent, and a one-hot encoding of the agents skill(s) (two skills if second-order). Lastly, this encoding is fed into a 3-layer MLP.

For crafting, we additionally concatenate a one-hot encoding of the task sketch (a list of the subgoals needed to complete the task).
A.6. Crafting Task Results

Figure 5. Episode returns for each task of the HRL agent on Crafting after training the low level skills for 10M environment steps. We plotted the mean and standard deviations over 8 seeds.