DHRL: A Graph-Based Approach for Long-Horizon and Sparse Hierarchical Reinforcement Learning

Seungjae Lee[1,2], **Jigang Kim**[1,2], **Inkyu Jang**[1,3], **H. Jin Kim**[1,3]
[1]Seoul National University
[2]Artificial Intelligence Institute of Seoul National University (AIIS)
[3]Automation and Systems Research Institute (ASRI)
{ysz0301, jgkim2020, leplusbon, hjinkim}@snu.ac.kr

Abstract

Hierarchical Reinforcement Learning (HRL) has made notable progress in complex control tasks by leveraging temporal abstraction. However, previous HRL algorithms often suffer from serious data inefficiency as environments get large. The extended components, *i.e.*, goal space and length of episodes, impose a burden on either one or both high-level and low-level policies since both levels share the total horizon of the episode. In this paper, we present a method of **D**ecoupling **H**orizons Using a Graph in Hierarchical **R**einforcement **L**earning (DHRL) which can alleviate this problem by decoupling the horizons of high-level and low-level policies and bridging the gap between the length of both horizons using a graph. DHRL provides a freely stretchable high-level action interval, which facilitates longer temporal abstraction and faster training in complex tasks. Our method outperforms state-of-the-art HRL algorithms in typical HRL environments. Moreover, DHRL achieves long and complex locomotion and manipulation tasks.

1 Introduction

Reinforcement Learning (RL) has been successfully applied to a range of robot systems, such as locomotion tasks [23, 7], learning to control aerial robots [9, 12], and robot manipulation [13, 21]. Goal-conditioned RL, which augments state with the goal to train an agent for various goals [22, 18], further raised the applicability of RL in robot systems allowing the agent to achieve diverse tasks.

Hierarchical Reinforcement Learning (HRL), which trains multiple levels of goal-conditioned RL, has improved the performance of RL in complex and sparse tasks with long horizons using temporally extended policy [24, 25, 15]. On the back of these strengths, HRL was adopted to solve various complex robotics tasks [19, 10, 17].

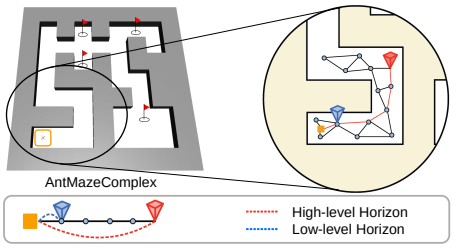

Figure 1: **DHRL:** By decoupling the horizons of both levels of the hierarchical network, DHRL not only solves long and sparse tasks but also significantly outperforms previous state-of-the-art algorithms.

However, HRL often has difficulty in complex or large environments because of training inefficiency. Previous studies speculated that the cause of this problem is the large goal space, and restricted the high-level action space to alleviate this phenomenon [27, 11]. Nevertheless, this approach performs well only in limited length and complexity and still suffers from the same trouble in larger environments.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

We show that this practical limitation of HRL can be mitigated by breaking down the coupled horizons of HRL. In previous HRL frameworks, the horizons of the low level and high level are related to each other structurally because they share the total length of the episode. This relation causes a tradeoff between the training burden of both levels; if the intervals between high-level actions increase (x-axis in Figure 2), the low-level policy has to cover a wider range, and in the opposite case, the high-level policy takes charge of the extended burden alone in large environments (y-axis in Figure 2). This is the reason why the previous HRL algorithms cannot cope with extended components of large environments (see Table 1 for the performance of the previous HRL method at various intervals).

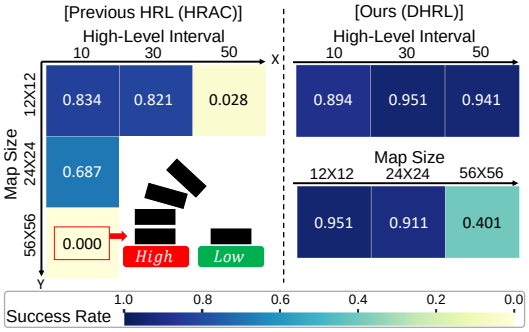

Figure 2: Our method is scalable in large environments by breaking down the relations between the two levels and allowing both levels to operate at their suitable horizons.

We break down the coupled horizons of HRL: To break the relation between the horizons of both levels, we adopt a graph structure. In our method, the high-level policy can use a longer temporal abstraction while the lower one only takes charge of smaller coverage by decomposing the subgoal into several waypoints with a graph. In this way, the HRL algorithm obtains the capability to stretch the interval of high-level action freely and achieves complex and large tasks, thanks to the enlarged strength of the HRL.

In summary, our main contributions are:

- We show that the previous HRL structures are not scalable in large environments, and that this limitation can be mitigated by removing coupled traits of high level and low level.
- To break down the coupled traits of HRL, we propose DHRL which decouples the horizons of high-level and low-level policies and bridges the gap using a graph.
- Our algorithm outperforms state-of-the-art algorithms in typical HRL environments and achieves complex and long tasks.

2 Preliminaries

We consider a finite-horizon Universal Markov Decision Process (UMDP) which can be represented as a tuple $(\mathcal{S}, \mathcal{G}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ where $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{G}$ are state space, goal space and action space respectively. The environment is defined by the transition distribution $\mathcal{T}(s_{t+1}|s_t, a_t)$ and reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$, where $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ are the state and action at timestep t respectively. Also, total return of a trajectory $\tau = (s_0, a_0, ..., s_H, a_H)$ is $R(\tau, g) = \sum_{t=0}^{H-1} \gamma^t r(s_{t+1}, g)$ where $r(s_{t+1}, g)$ (or $r(s_t, a_t, g)$) is a goal conditioned reward function and γ is a discount factor. Subgoal sg, waypoint wp and goal g are defined in goal space $\mathcal{G}$ and we consider $\mathcal{G}$ that is a subspace of $\mathcal{S}$ with a mapping $\psi : \mathcal{S} \to \mathcal{G}$.

HRL framework typically has high-level policy π^{hi} and low-level policy π^{lo}, each maintaining a separated replay buffer $\mathcal{B}^{\text{hi}}$ and $\mathcal{B}^{\text{lo}}$. Our method also follows the general HRL framework and employs both buffers that store high-level $(s_t, g_t, sg_t, r_t, s_{t+c_h}) \in \mathcal{B}^{\text{hi}}$ and low-level transition data $(s_t, wp_t, a_t, r_t, s_{t+1}) \in \mathcal{B}^{\text{lo}}$, where c_h is the interval between high-level action.

One of the key problems in learning HRL is that the low-level policy π^{lo} is non-stationary and thus old data from past policies may contain different next-states s_{t+c_h} even though the high-level policy provides the same subgoal in the identical state. To bypass this off-policy discrepancy, HRL models use off-policy correction which relabels the old subgoal of high-level policy to be the most 'plausible subgoal' that will result in a similar transition in old data with the current low-level agent [16, 27, 11] (HIRO-style off-policy correction). Other approaches propose to relabel high-level action sg_t to be achieved state s_{t+c_h} (HAC-style hindsight action relabelling), reducing the computation cost to find a 'plausible subgoal' [14].

Graph-guided RL methods, which combine the strength of RL and planning by decomposing a long-horizon task into multi-step sub-problems, estimate the temporal distance between states and goals to construct a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ on goal space $\mathcal{G}$ without additional prior knowledge about environments. Previous studies proposed various methods to recover distance from Q-value [3, 8, 26]. If the agent gets -1 reward at every step except when it is in a goal area where the agent gets 0 reward, then, $Q_{\text{lo}}(s, a|g)$ can reveal the temporal distance between s to g as: (Refer to the Appendix B for the detailed derivation.)

$$Dist(s \rightarrow g) = \log_{\gamma}\left(1 + (1-\gamma)Q_{\text{lo}}(s, \pi(s, g)|g)\right) \tag{1}$$

However, it is known that recovering temporal distance correctly from the vanilla Q-network in this setting is challenging. For that reason, the previous methods use an additional Value function approximator [26] or distributional Q-networks [3].

3 Related work

Graph-guided RL. Graphs have recently been used as a non-parametric model in reinforcement learning (RL) to combine the advantages of RL and planning [3, 8, 26, 2, 5]. By decomposing a long-horizon task into multi-step planning problems, these studies have shown better performance and data efficiency. Search on the Replay Buffer (SORB) [3] constructs a directed graph based on the states randomly extracted from a replay buffer and Q-function-based edge cost estimation. The follow-up studies further improved the performance of earlier graph-guided RLs [3, 8] by combining additional methods such as graph search on latent space[26] or model predictive control [2].

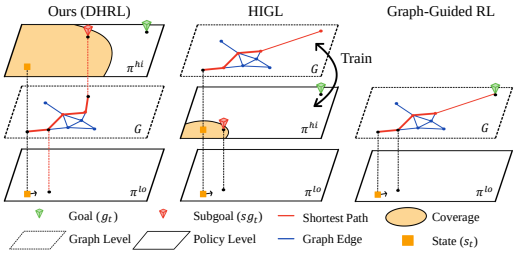

Figure 3: The differences between DHRL and the previous graph-based HRL methods. Our algorithm includes the graph structure between both levels explicitly while the previous methods use the graph only for training high-level policy or getting waypoints.

However, previous papers sidestepped the exploration problems in complex tasks through 'uniform initial state distribution' [3, 8, 26, 5], or work only on the dense reward settings [2]. We emphasize that the 'uniform initial state distribution' accesses privileged information about the environment during training by generating the agent uniformly within the feasible area of the map. This greatly reduces the scope of application of these algorithms. Unlike prior methods, ours can train from sparse reward settings and a 'fixed initial state distribution' without knowledge of the agent's surroundings, which makes it practical for physical settings. For detailed examples and comparison of various initial state distributions, see Table 2 and Figure 11 in Appendix C.

Constrained-subgoal HRL. To mitigate the training inefficiency issue of HRL, several researchers proposed methods that restrict the action of high-level agent to be placed in adjacent areas. Hierarchical Reinforcement Learning with k-step Adjacency Constraint (HRAC) [27] limits the subgoal to be in the adjacency space of the current state. Hierarchical reinforcement learning Guided by Landmarks (HIGL) [11] improved the data efficiency by adding novelty-based landmarks to adjacency-constrained HRL. However, these improvements only work on the limited length and complexity of the environment.

The previous work closest to our method is HIGL. However, there are three key differences between the previous work and our approach. First, our model explicitly includes the whole graph structure while HIGL needs to train high-level action to imitate the graph by using an additional loss term corresponding to the Euclidean distance from the nodes of the graph. Second, we use a graph to decouple the horizons of high level and low level, unlike the previous method which uses the graph only for guidance. Most importantly, ours can achieve goals in long and complex environments. To the best of our knowledge, there is no prior HRL research to train a model that has decoupled the horizons of the two levels.

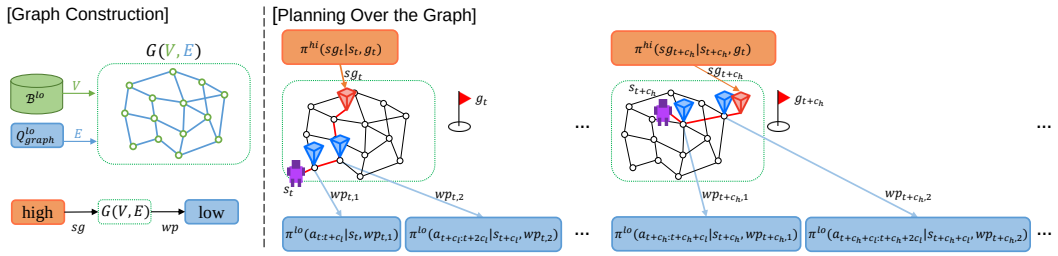

Figure 4: An overview of DHRL which includes the mid-level non-parametric policy between the high and low levels. The high-level (orange box) policy delivers subgoal $sg \in \mathcal{G}$ to the graph level (green box) and the graph instructs the low-level policy (blue box) to reach the waypoint $wp \in \mathcal{G}$. c_h represents how long each high-level action operates for. The low level is given $c_{l,i}$ steps to achieve the goal where $c_h \neq c_{l,i}$.

4 Methods

We introduce Decoupling Horizons Using a Graph in Hierarchical Reinforcement Learning (DHRL), which can separate the time horizons of high-level and low-level policies and bridge the gap between both horizons using a graph. Our framework consists of high-level policy $\pi^{\text{hi}}(sg|s, g)$, low-level policy $\pi^{\text{lo}}(a|s, wp)$, and a graph $\mathbf{G}$. Given a goal g_t in the environment, the high-level policy outputs a subgoal sg_t (see Figure 4). Then, the shortest path from the current state s_t to sg_t is found on the graph. To do so, s_t and sg_t are added to the existing graph structure, then a sequence of waypoints $(s_t, wp_{t,1}, wp_{t,2}, ..., sg_t)$ is returned using a graph search algorithm. Finally, the low-level policy tries to achieve $wp_{t,i}$ during $c_{l,i}$ steps.

The key point of our method is that the low-level horizon $h^{\text{low}} = c_l$ is unrelated to the high-level horizon h^{high}. Since the high level generates one subgoal every c_h steps, the H-step task is a H/c_h-step task for a high-level agent ($h^{\text{high}} = H/c_h$) where c_h is the interval between high-level action ($c_h > c_l$). In other words, unlike the previous HRL methods which have the relationship of

$$h^{\text{high}} \times h^{\text{low}} = H, \tag{2}$$

our algorithm does not have such relations, removing an obstacle toward a scalable-RL algorithm. Since the low-level horizon c_l is determined by the edge cost between waypoints, we can also express the c_l between the i-1th waypoint and i^{th} waypoint as $c_{l,i}$, but we omit the letter i in the later statements that do not need to specify the waypoint.

In section 4.1, we explain how to construct a graph over states and find a path on the graph level. In section 4.2, we present a low-level policy which can recover temporal distance between states without overestimation. In section 4.3, We introduce a strategy to train our method through graph-agnostic off-policy learning. Finally, in section 4.4, we propose additional techniques for better data efficiency in large environments.

4.1 Graph level: planning over the graph

This section details the graph search part in DHRL planning. We emphasize that every distance in the DHRL model is based on temporal distance $Dist(\cdot \rightarrow \cdot)$, obtained through Eq. (1). Thus, our algorithm requires no further information about the environment ($e.g.$ Euclidean distance between states) than general HRL settings.

To find the shortest path, we adopt Dijkstra's Algorithm, as in the previous study [3]. The differences from the previous methods [3, 26] are the existence of the high-level policy, and whether the secondary path is considered. Let $\psi : \mathcal{S} \rightarrow \mathcal{G}$ be the projection of states on the goal space. In the graph initialization phase, we samples n nodes (also called landmarks) using FPS algorithm (Algorithm 2 in Appendix A) from $\psi(s)$ where $s \in \mathcal{S}$ is state sampled from $\mathcal{B}^{\text{lo}}$. Then, we connect a directed edge $s_1 \rightarrow s_2$ if the temporal distance from s_1 to s_2 is less than the cutoff-threshold. In the planning phase, the graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$ gets the subgoal sg_t from the high-level policy and adds the projection of current state $\psi(s_t)$ and sg_t to $\mathbf{V}$, so that the number of nodes in $\mathbf{G}$ becomes $n + 2$. Then, we connect the edges with costs less than the cutoff-threshold between the newly added nodes

and existing nodes. Next, we find the sequence of waypoints $\mathcal{W} : (wp_{t,0} = \psi(s_t), wp_{t,1}, wp_{t,2}, ..., wp_{t,k-1}, wp_{t,k} = sg_t)$ that connects from $\psi(s_t)$ to sg_t using a graph search algorithm. At this time, if there is no path from $\psi(s_t)$ to sg_t, we adopt a secondary path from $\psi(s_t)$ to $sg_t^\dagger$, where $sg_t^\dagger$ is the closest node to sg_t among the nodes connected from $\psi(s_t)$. After finding the waypoint sequence, the graph level provides $wp_{t,1}$ to the low-level policy and instructs it to reach $wp_{t,1}$. If it has been $Dist(wp_{t,i-1} \rightarrow wp_{t,i})$ steps since the agent started tracking $wp_{t,i}$ or it achieves $wp_{t,i}$, then we update the current tracking waypoint to the next waypoint $wp_{t,i+1}$. In order to reflect the newly discovered state, we update graph nodes at every N episodes.

However, it is challenging to train Q-network which recovers reliable temporal distance. We explain the reason and a solution in the next section.

4.2 Low level: Separate Q-networks for Graph and Critic (SQGC)

Unlike previous graph-guided RL methods that leverage a graph only after the actor and critic networks have been learned, we construct and utilize a graph during training. To do so, the low-level policy Q^{lo} is evaluated during training to assign edge costs. However, when π^{lo} is not yet competent in achieving some goals or encounters difficult goals, the accumulation of failure experiences in the replay buffer causes underestimation of Q-values. This leads to an overestimation of temporal distance reconstructed from Eq. (1) and spoils the graph near the overestimated region by making the node-selection algorithm (Algorithm 2 in Appendix A) select more and more graph nodes around the overestimated area. Therefore, a temporal-distance reconstruction method is needed even when the policy is not sufficiently trained.

For this reason, we propose Separate Q-networks for Graph and Critic (SQGC) to prevent temporal distance overestimation. SQGC is composed of two identical Q-networks using different proportions of hindsight goal relabeling (HER) [1]. The SQGC includes $Q^{\text{lo}}_{\text{critic}}$ and $Q^{\text{lo}}_{\text{graph}}$ where $Q^{\text{lo}}_{\text{critic}}$ is for training π^{lo}, just like a typical application, and $Q^{\text{lo}}_{\text{graph}}$ is for recovering temporal distance between nodes. We substitute wp_t with $\hat{wp}_t := ag_{t+t_{\text{ftr}}}$ in the sequential transition of a single episode $(s_t, wp_t, a_t, r(s_{t+1}, wp_t), s_{t+1})_{t=1:H-1}$ where ag means the achieved goal and t_{ftr} is a random integer drawn from the uniform distribution between 0 and $H - t$. To train $Q^{\text{lo}}_{\text{graph}}$, we relabel 100% of wp_t in $(s_t, wp_t, a_t, r(s_{t+1}, wp_t), s_{t+1})_{t=1:H-1}$ as $\hat{wp}_t$, while we replace only 80% of wp_t for $Q^{\text{lo}}_{\text{critic}}$.

Our method can prevent overestimation in the distance recovery by relabelling the goals in all transitions for training $Q^{\text{lo}}_{\text{graph}}$ because the experiences of failure are replaced with the successful trajectory (see section 5.3 for the ablation study). Also, by maintaining the original $Q^{\text{critic}}_{\text{lo}}$ to train π_{lo}, there is no degradation in the performance of the agent who might otherwise not be able to get negative feedback from failure since the failure will be relabeled as a desired goal.

However, it is still challenging to train HRL including a graph level using off-policy RL algorithms. We describe our approach to train DHRL using an off-policy algorithm in the next section.

4.3 High level: hindsight transitions for graph-agnostic off-policy learning

Thanks to decoupling the time horizons of both levels in HRL, the high-level policy in our method can look further without any additional burden on the low-level policy. In other words, We can stretch c_h, which represents how long each high-level action operates for. However, because of the extended interval and non-stationarity of the high-level MDP, it is challenging to train DHRL with an off-policy algorithm, which is important for data efficiency in that the off-policy algorithm can use the previous data from the replay buffer.

This non-stationarity of the high-level MDP is caused by the presence of a graph and low-level policy. Since the graph is gradually updated, it is challenging to train a model using an off-policy method from the data given by the previous graph. Furthermore, as c_h gets longer, predicting the similarity of the trajectory of c_h steps with only the first step of $\pi^{\text{lo}}(a_i|s_i, g_i)$ gets more difficult when we replace subgoal using HIRO-style off-policy correction [16] which is a popular method adopted by SOTA HRL algorithms [27, 11].

To facilitate the off-policy learning for longer c_h and the changing graph in DHRL, we adopt a well-known hindsight action relabelling method proposed in Hierarchical Actor-Critic (HAC)

[14] in the high-level replay buffer data $(s_t, g_t, sg_t, r_t, s_{t+c_h}) \in \mathcal{B}^{\text{hi}}$. Alongside the original transitions data, we copy the transition data and replace the subgoal ($sg_t = \pi_\beta^{\text{high}}(sg_t|s_t, g_t)$) with the achieved goal ag_{t+c_h} after c_h steps. Thus, we use both transitions $(s_t, g_t, sg_t, r_t, s_{t+c_h})$ and $(s_t, g_t, ag_{t+c_h}, r_t, s_{t+c_h})$ to train the high-level policy. Our method utilizes the optimality of the graph and low-level policy to include a graph in the learning process, while the HAC-style hindsight action relabelling method assumes the optimal low-level policy $\pi^{\text{lo}*}$ only.

By replacing the previous subgoal with the achieved goal, we can assume that this transition was obtained from a stationary graph and $\pi^{\text{lo}*}$ with an error below a bound which is set to be a function of the density of the graph. We provide a theoretical analysis of the possibility of replacing the old off-policy graph with a virtual stationary graph in this section and Appendix B.

Definition 4.1. Given a compact state space $\mathcal{S}$, $\mathbf{G}(\mathbf{V}, \mathbf{E})$ is an $\epsilon-$resolution graph $if\ \forall s \in \mathcal{S}, \exists v \in V$ $s.t. \max(Dist(\psi(s) \to v), Dist(v \to \psi(s))) < \epsilon$, where $\mathbf{V} \subset \psi(\mathcal{S})$ and $E = \{(v_i, v_j)|v_i, v_j \in \mathbf{V}, Dist(v_i \to v_j) < c_l\}$.

Theorem 4.2. *Let $\mathbf{G}$ be an arbitrary $\epsilon-$resolution graph ($\epsilon < c_l/2$). Also, let π_β^{lo} and $\mathbf{G}_\beta$ be the low-level policy and graph at the time of collecting the data. Off-policy error rate $\rho(\mathbf{G})$ is the normalized distance error with respect to the total traversal distance according to the change of π_β^{lo} and $\mathbf{G}_\beta$ to $\pi^{\text{lo}*}$ and $\mathbf{G}$. If there is a path from s to g, the upper bound of off-policy error rate $\rho(\mathbf{G})$ using a path obtained from graph search over $\mathbf{G}$ is $2\epsilon/c_l$.*

Theorem 4.2 shows that if we replace the previous subgoal with the achieved goal, and assume that these transitions are obtained from a stationary $\epsilon-$resolution graph $\mathbf{G}$, then the off-policy error ρ is less than $2\epsilon/c_l$. Thus, we can train DHRL through a graph-agnostic off-policy RL algorithm using the substituted transition.

4.4 Optional techniques: gradual penalty and frontier-based goal-shifting

In this section, we propose two additional techniques, gradual penalty and frontier-based goal-shifting (FGS). These optional techniques can boost the performance of DHRL on some long tasks as shown in Figure 9(a) and (c). We note that these are not essential to train DHRL and ours outperforms previous HRL frameworks without these techniques (see section 5.3 for the ablation study).

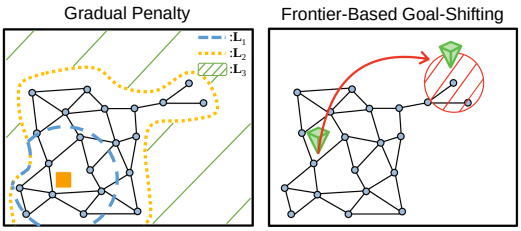

Figure 5: Optional techniques for better data efficiency: gradual penalty encourages the high-level policy to output a subgoal that the low-level agent can achieve. Frontier-based goal-shifting replaces goals in the explored area with new goals positioned in the rim of the graph.

Gradual penalty. Similar to the subgoal testing in the previous method [14], we propose gradual penalty (algorithm 4 in Appendix A), which can impose the penalty more delicately when the action space of the high-level agent gets large. We evaluate sg_t of the original transition data $(s_t, g_t, sg_t, r_t, s_{t+c_h})$ by categorizing the following three cases; (a) close to the graph and low level actually achieved the subgoal $\in \mathbf{L}_1$, (b) close to the graph but the low-level policy could not achieve the subgoal $\in \mathbf{L}_2$ and (c) far from the graph $\in \mathbf{L}_3$. In this way, we can impose the more detailed penalties to respond to the expansion of the high-level action space as the high-level interval c_h stretches.

Frontier-based goal-shifting. This optional technique has been devised to accelerate learning in a complex environment (algorithm 5 in Appendix A). FGS moves the final goal g to the frontier area when g comes into the place where the graph is already laid out during training. To check whether the goal is in the graph area, we examine whether $\min_{v \in \mathbf{V}}(Dist(v \to g))$ is smaller than cut-off threshold, where $\mathbf{V}$ is the set of the graph nodes. Alternative goals are the addition of random noise to nodes sampled from $v \in \mathbf{V}$ proportional to $-Q(s_0, \pi(s_0, v)|v)$. FGS is similar to the previous goal-directed exploration research [20] in that both use weighted samples, but ours does not maintain a generative model and samples the goals from the graph level in DHRL. Note that we do not use FGS when we compare DHRL with the previous state-of-the-art, since this FGS is beyond the main contribution, which is about HRL structure.

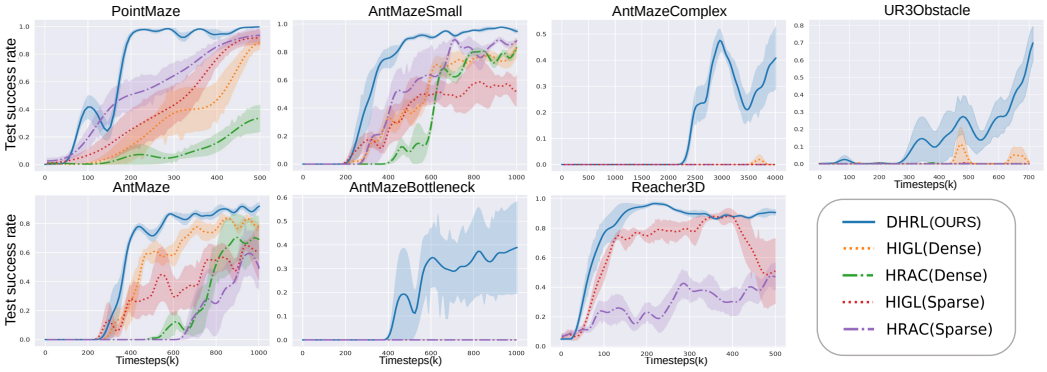

Figure 6: DHRL significantly outperforms prior state-of-the-art algorithms (success rate averaged over 4 random seeds and smoothed equally, and only the sparse settings for Reacher3D are tested as in the previous papers). Note that in AntMazeComplex, AntMazeBottleneck, and UR3Obstacle, the curves are not visible as they overlap at zero success rate.

5 Experiments

5.1 Environment description

We evaluate DHRL on robot environments based on the MuJoCo simulator including some sparse and long-horizon tasks. Firstly, various locomotion environments with 'fixed initial state distributions' are used to validate the temporal abstraction capability of DHRL in long-horizon and cluttered environments. In test episodes of the locomotion environments, the agent gets one of the most challenging goals ($i.e.$, the end of the maze). We also evaluate our algorithm in a robot arm environment in which the agent aims to make the end-effector touch the goal, to evaluate our method in more complex dynamics.

- PointMaze / AntMazeSmall : The point / ant achieves the goal if it comes within 2.5 distance (success threshold) from the target point in 12×12 maze.
- AntMaze : 24×24 maze with success threshold 5
- AntMazeBottleneck : Bottleneck exists at the middle of the maze. The ant can barely pass through bottleneck.
- AntMazeComplex : 56×56 maze with success threshold 5
- Reacher3D : 7-dof robot arm aims to reach a goal.
- UR3Obstacle : 6-dof robot arm aims to reach a goal in an environment with several board-shaped obstacles.

We adopt TD3 algorithm [4] for high-level and low-level networks and use Dijkstra algorithm to find the shortest path in the graph level. Also, we note that only a few hyperparameters have been changed across various locomotion experiments: the number of nodes, penalty, and c_h. The results of DHRL in this paper are obtained using only sparse reward.

5.2 Experiment result

Baselines. We compare our method with state-of-the-art algorithms with and without a graph respectively; HIGL [11] and HRAC [27]. For more comparison with shallow RL (SAC) [6] and vanilla HRL (HIRO) [16], see Table 3 in Appendix C.

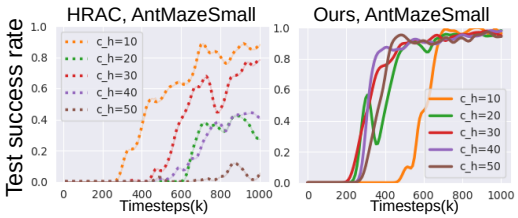

Figure 7: Robustness to c_h: HRAC vs Ours

Comparison to state-of-the-art algorithms. Results are shown in Figure 6. Thanks to decoupling the horizons of the levels, DHRL could stretch the high-level interval and shows high data efficiency and success rate in various locomotion

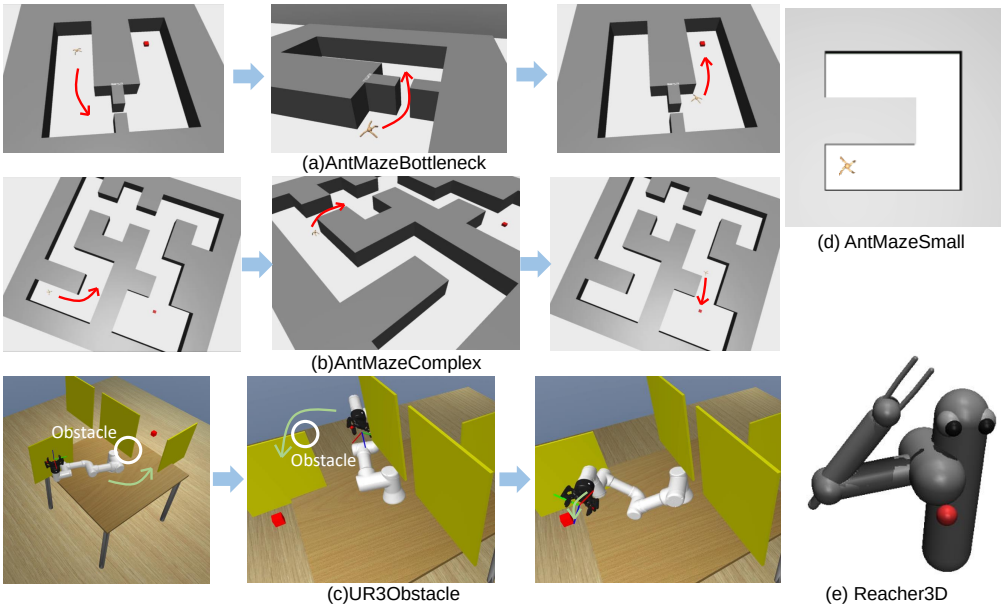

Figure 8: Locomotion and Reacher tasks in simulation: Additional to general tasks for HRL, we evaluated our method in long and sparse tasks (see (a), (b), and (c) in the figure). The maps of the environments are not given to the agent.

Table 1: The tradeoff in performance between the high level and the low level.

SUCCESS RATE	HRAC $c_h = 5$	HRAC $c_h = 10$	HRAC $c_h = 30$	HRAC $c_h = 50$	**DHRL**
12×12 MAP	43.0%	**88.4%**	78.3%	4.5%	**95.1%**
24×24 MAP	18.0%	48.9%	**57.4%**	16.4%	**91.1%**
56×56 MAP	0.0%	0.0%	0.0%	0.0%	**40.1%**

and goal-reaching tasks. Moreover, DHRL is the only algorithm that can succeed in complex environments (AntMazeComplex and AntMazeBottleneck). When measuring the performance of the baselines, we selected c_h with the best performance among the values c_h=10, 20, ..., and 80.

As shown in Figure 6, previous HRL methods cannot solve long-horizon tasks. This is likely due to the coupling ($c_h = c_l$) of the high-level horizon ($h^{\text{high}} = H/c_h$) and the low-level horizon ($h^{\text{low}} = c_l$) resulting in an increased burden on either the high-level or the low-level policy. For long-horizon tasks (large H), if $c_h(= c_l)$ is fixed, then h^{high} increases and the high-level performance plunges (Table 1). On the other hand, if $c_h(= c_l)$ is increased, the low level has to manage a wider area and the performance plunges as shown in Figure 7 and Table 1.

5.3 Ablation study

Decoupling horizons. In this section, we examine how decoupling horizons affects the performance of long-horizon HRL and evaluate whether DHRL can stretch the interval of high-level policy. As shown in Figure 9(b), we tested various values of spacing of the high-level action, c_h. The result shows that even in long intervals of the high-level policy, DHRL shows consistent or improved performance (Figure 9(b)) without major degradation unlike previous HRL methods (Figure 7). In particular, DHRL agent in long-horizon environments such as AntMazeComplex shows that extended high-level interval is crucial for exploration and performance (see the rightmost figure in Figure 9(b)). Considering that the baselines could not solve long-horizon tasks even with various values of c_h, and the performance degrades with increasing c_h, we conclude that our method successfully decoupled the time steps in both levels and stretched the interval between high-level actions. This trait allows the high-level policy to look further and take advantage of the extended temporal abstraction.

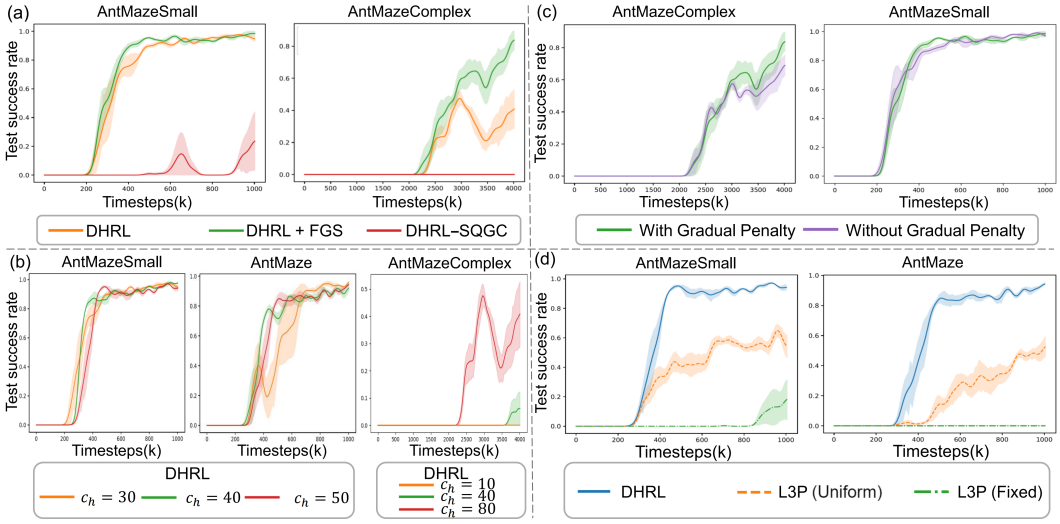

Figure 9: **Ablation study.** (a): The separate Q-network (SQGC) effectively alleviates the overestimation problem of temporal distance and FGS improves data efficiency. (b): Extending the high-level horizon is crucial in long-horizon tasks and DHRL successfully separates the horizons. (c): Gradual penalty improves the data efficiency in long tasks. (d): Without an additional high-level policy above the graph level, the performance plunges without the 'uniform initial state distribution'.

Separate Q-network for Graph and Critic (SQGC). In this section, we evaluate the effects of the Separate Q-network for Graph and Critic (SQGC) in our method. The red line in Figure 9(a), which is a variant of DHRL without an SQGC, clearly shows that the SQGC is critical to the performance. We empirically found that the overestimation of the temporal distance between states occurs especially near the obstacles ($e.g.$ corners at the maze), and it is difficult for an agent to pass by without a separate Q network for graph construction. This is consistent with our analysis that the experiences of failure spoil the ability to recover the temporal distance from Q-network.

Ablate high-level policy. Graph-guided RL, which maintains graph level and low-level policy, is also a variant of HRL in that the graph level is a non-parametric version of the high-level policy (see Figure 3). From this point of view, the main difference between graph-guided RL and our method is the existence of the high-level policy. Then, why do we need additional high-level policy above the graph level?

Most graph-guided RL algorithms use 'uniform initial state distribution' to train the agent in complex environments and such assumption could be expensive, especially in the physical world where it is challenging to start from different positions each time. Figure 9(d) shows the result of the graph-guided RL algorithm, L3P [26], with a uniform and fixed initial state distributions. This indicates that without 'uniform initial state distribution', the performance and data efficiency of the previous graph-guided RL method drops seriously even in the smallest environment we experimented with. In contrast, DHRL can explore without 'uniform initial state distribution' even in long environments thanks to the enlarged temporal abstraction and exploration performance. Considering that the high-level policy suggests a subgoal to graph level, we conclude that the high level facilitates better exploration of the graph-guided RL. Thus, DHRL that has exploration capability also can be seen as the improved version of graph-guided RL.

Optional techniques. The result in Figure 9(c) shows that the gradual penalty can improve data efficiency in long tasks. By imposing a penalty on subgoals that are far from the current graph, the gradual penalty encourages the high-level agent to output a subgoal near the graph. Since the low-level agent can hardly achieve subgoals far from the explored area, the gradual penalty accelerates the training by providing more valid goals. FGS also can improve training efficiency in complex environments (Figure 9(a)). This means that it is important to provide goals near the unexplored areas which help the agent explore the unseen states in complex environments. With FGS, DHRL efficiently guides the agent by sampling the goals located on the rim of the graph.

6 Conclusion

We proposed a Decoupling Horizons Using a Graph in Hierarchical Reinforcement Learning (DHRL), which is a data-efficient HRL algorithm leveraging a graph to expand the range of problems that HRL can solve, by decoupling horizons and allowing both levels to operate at their suitable horizons. Our experimental results show that our method successfully separates the horizons of the levels and outperforms prior state-of-the-art methods. We believe that our method can present a direction towards scalable HRL allowing the hierarchical policy to take advantage of the extended temporal abstraction and have a lower dependency on the horizon of the environment. In this paper, we adopted a vanilla graph construction algorithm and sparse reward settings. We expect that if a novelty-based node selection is added to the graph construction, the performance could be further improved.

Limitation. While our algorithm shows successful results in complex and long-horizon tasks, it might be difficult to construct a graph in some tasks that require complex interactions with the environment because of the higher-dimensional state space. We expect that this limitation can be solved by extending our algorithms into latent state space, or by incorporating a node sparsification algorithm, which we leave for future works.

7 Acknowledgement

This work was supported by AI based Flight Control Research Laboratory funded by Defense Acquisition Program Administration under Grant UD200045CD. Also, Seungjae Lee is grateful for financial support from Hyundai Motor Chung Mong-Koo Foundation.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5055–5065, 2017.

[2] Akhil Bagaria, Jason K Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning*, pages 521–531. PMLR, 2021.

[3] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[4] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[5] Robert Gieselmann and Florian T Pokorny. Planning-augmented hierarchical reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):5097–5104, 2021.

[6] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[7] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. In *Robotics: Science and Systems*, 2019.

[8] Zhiao Huang, Fangchen Liu, and Hao Su. Mapping state space using landmarks for universal goal reaching. *Advances in Neural Information Processing Systems*, 32:1942–1952, 2019.

[9] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[10] Deepali Jain, Atil Iscen, and Ken Caluwaerts. Hierarchical reinforcement learning for quadruped locomotion. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7551–7557. IEEE, 2019.

[11] Junsu Kim, Younggyo Seo, and Jinwoo Shin. Landmark-guided subgoal generation in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[12] Nathan O Lambert, Daniel S Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer SJ Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

[13] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[14] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *Proceedings of International Conference on Learning Representations*, 2019.

[15] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. In *International Conference on Learning Representations*, 2018.

[16] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 31:3303–3313, 2018.

[17] Ofir Nachum, Michael Ahn, Hugo Ponte, Shixiang Shane Gu, and Vikash Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. In *Conference on Robot Learning*, pages 110–121. PMLR, 2020.

[18] Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. *Advances in Neural Information Processing Systems*, 32:14843–14854, 2019.

[19] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

[20] Vitchyr Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *International Conference on Machine Learning*, pages 7783–7792. PMLR, 2020.

[21] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[22] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

[23] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/schulman15.html`.

[24] Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic attentive writer for learning macro-actions. *Advances in Neural Information Processing Systems*, 29:3486–3494, 2016.

[25] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017.

[26] Lunjun Zhang, Ge Yang, and Bradly C Stadie. World model as a graph: Learning latent landmarks for planning. In *International Conference on Machine Learning*, pages 12611–12620. PMLR, 2021.

[27] Tianren Zhang, Shangqi Guo, Tian Tan, Xiaolin Hu, and Feng Chen. Generating adjacency-constrained subgoals in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6
   (c) Did you discuss any potential negative societal impacts of your work? [No] This work inherits the potential negative social impact of reinforcement learning in robotics. We do not anticipate that there will be any additional negative impact of this paper.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section B in Appendix
   (b) Did you include complete proofs of all theoretical results? [Yes] See Section B in Appendix
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplemental material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Figure 6
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section C in Appendix
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 5.1
   (b) Did you mention the license of the assets? [Yes] See Section 5.1
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Algorithms.

Algorithm 1 Training DHRL

sample $D_i^{\text{lo}} = (s_t, wp_t, a_t, r(s_{t+1}, wp_t), s_{t+1})_i \in \mathcal{B}^{\text{lo}}$
relabel $wp_t \leftarrow \hat{wp}_t = ag_{t+t_{\text{ftr}}}$ to make $\hat{D}_i^{\text{lo}}$
update $Q^{\text{lo}}_{\text{critic},\theta_1}$ and $\pi^{\text{lo}}_{\phi_1}$ using $D_i^{\text{lo}} \cup \hat{D}_i^{\text{lo}}$
update $Q^{\text{lo}}_{\text{graph},\theta_2}$ using $\hat{D}_i^{\text{lo}}$
if t mod d **then**
  sample $D_i^{\text{hi}} = (s_t, g_t, sg_t, r_t, s_{t+c_h})_i \in \mathcal{B}^{\text{hi}}$
  relabel $sg_t \leftarrow ag_{t+c_h}$ to make $\hat{D}_i^{\text{hi}}$
  for $(s_t, g_t, sg_t, r_t, s_{t+c_h})$ in D_i^{hi} **do**
    if $r(s_{t+c_h}, sg_t)$ = 0 **then**
      $r_t \leftarrow r_t$ $\lhd$ subgoal $\in \mathbf{L}_1$
    else if use GradualPenalty **then**
      $r_t \leftarrow$ `GradualPenalty`(graph $\mathbf{G}$, sg_t, $Q^{\text{lo}}_{\text{graph}}$)
    else
      $r_t \leftarrow$ penalty p_1
    end if
  end for
  update $Q^{\text{hi}}_{\theta_3}$ and $\pi^{\text{hi}}_{\phi_2}$ using $D_i^{\text{hi}} \cup \hat{D}_i^{\text{hi}}$
end if

Algorithm 2 Farthest Point Sampling Algorithm

Input: set of states $\{s_1, s_2, ..s_K\}$, sampling number k, temporal distance function $Dist(\cdot \rightarrow \cdot)$
`SelectedNode` = []
`DistList` = [inf, inf, ... inf]
for $i = 1$ **to** k **do**
  `FarthestNode` $\leftarrow$ argmax(`DistList`)
  add `FarthestNode` to `SelectedNode`
  `DistFromFarthest` $\leftarrow [Dist(\texttt{FarthestNode} \rightarrow s_1), ..., Dist(\texttt{FarthestNode} \rightarrow s_K)]$
  `DistList` = ElementwiseMin(`DistFromFarthest`, `DistList`)
end for
return `SelectedNode`

Algorithm 3 Planning with DHRL

while not done **do**
  if t mod $graph_construct_freq$ **then**
    construct a graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$: sample $\mathbf{V} = \psi(s)$ where $s \in D^{\text{lo}}$ through FPS algorithm and get edge cost $\mathbf{E}$ by Eq. (1)
  end if
  sg_t = $\pi^{\text{hi}}(s_t, g_t)$
  get $\mathcal{W} : (wp_{t,0} = \psi(s_t), wp_{t,1}, wp_{t,2}, ..., wp_{t,k-1}, wp_{t,k} = sg_t)$
  previous waypoint index $idp \leftarrow 0$; tracking waypoint index $idt \leftarrow 1$; tracking time $t_{tr} \leftarrow 0$
  for $\tau = 1$ **to** c_h **do**
    get low-level action a_τ from $\pi^{\text{lo}}(a_\tau | s_\tau, wp_{t,idt})$
    act a_τ in the environment and get $s_{\tau+1}$
    t_{tr} += 1; t += 1
    if agent achieve $wp_{t,idt}$ or $t^{tr} > Dist(wp_{t,idp} \rightarrow wp_{t,idt})$ **then**
      idp += 1; idt += 1; $t_{tr} \leftarrow 0$
    end if
  end for
end while

Algorithm 4 Gradual Penalty

Input: graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$, subgoal sg_t, $Q^{\text{lo}}_{\text{graph},\theta_2}$, gradual penalty threshold ζ_1, penalty p_1, penalty p_2
if $\min(Q^{\text{lo}}_{\text{graph},\theta_2}(v \in \mathbf{V}, sg_t)) < \zeta_1$ **then**
  $r_t \leftarrow$ penalty p_1 $\triangleleft$ subgoal $\in \mathbf{L}_2$
else
  $r_t \leftarrow$ penalty p_2 $\triangleleft$ subgoal $\in \mathbf{L}_3$
end if

Algorithm 5 Frontier-Based Goal-Shifting (FGS)

Input: s_t, graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$, goal g, $Q^{\text{lo}}_{\text{graph},\theta_2}$, cut-off threshold ζ_2
$Dist(s, g) := \log_\gamma (1 + (1 - \gamma) Q^{\text{lo}}_{\text{graph},\theta_2}(s, \pi(s, g)|g))$
if $\min_{v \in \mathbf{V}}(Dist(v \to g)) < \zeta_2$ **then**
  $\mathbf{V}_{\text{candidate}} \leftarrow \mathbf{V} + \text{noise}$
  $g_t \leftarrow \texttt{random.choice}(\mathbf{V}_{\text{candidate}}, \text{weight} = -Q^{\text{lo}}_{\text{graph},\theta_2}(s_t, \pi(s_t, \mathbf{V}_{\text{candidate}})|\mathbf{V}_{\text{candidate}}))$
end if
return g_t

Algorithm 6 Overview of DHRL

Input: initial random steps $\tau_{\text{randomwalk}}$, initial steps without planning $\tau_{\text{w/o graph}}$, total training step τ_{total}, Env, low-level agent $Q^{\text{lo}}_{\text{critic},\theta_1}$, $Q^{\text{lo}}_{\text{graph},\theta_2}$ and $\pi^{\text{lo}}_{\phi_1}$, high-level agent $Q^{\text{hi}}_{\theta_3}$ and $\pi^{\text{hi}}_{\phi_2}$
$Dist(s, g) := \log_\gamma (1 + (1 - \gamma) Q^{\text{lo}}_{\text{graph},\theta_2}(s, \pi(s, g)|g))$
for $\tau = 1$ **to** τ_{total} **do**
  if Env.done **then**
    Env.reset (episode step resets to 0)
    if Use FGS **then**
      $g \leftarrow \text{FGS}(\mathbf{G}, g, Q^{\text{lo}}_{\text{graph},\theta_2})$
    end if
  end if
  if $\tau < \tau_{\text{randomwalk}}$ **then**
    $a_t \leftarrow$ random.uniform(high = action.high, low = action.low) $\triangleleft$ random action
  else if $\tau < \tau_{\text{w/o graph}}$ **then**
    $a_t \leftarrow$ vanilla $HRL(sg_t = \pi^{\text{hi}}_{\phi_2}(s_t, g)$ and $\pi^{\text{lo}}_{\phi_1}(s_t, sg_t))$ $\triangleleft$ act without planning
  else
    if Graph $\mathbf{G}$ is not initialized **then**
      Create a graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$ using FPS algorithm $\triangleleft$ initialize graph
    end if
    if episode step(the step of the environment) $\%$ c_l = 0 **then**
      $sg_t \leftarrow \pi^{\text{hi}}_{\phi_2}(s_t, g)$ $\triangleleft$ get subgoal
      $\{wp_{t,1}, wp_{t,2}, \cdots wp_{t,k}\} \leftarrow \texttt{Dijkstra'salgorithm}(s_t, sg_t)$ $\triangleleft$ get waypoints
      current waypoint index $n = 1$
    end if
    if achieved $wp_{t,n}$ or tried more than $Dist(wp_{t,n-1}, wp_{t,n})$ to achieve $wp_{t,n}$ **then**
      current waypoint index $+= 1$
    end if
    $a_t \leftarrow \pi^{\text{lo}}_{\phi_1}(s_t, wp_{t,n+1})$ $\triangleleft$ get low-level action
  end if
  Env.step(a_t)
  Train low-level agent $Q^{\text{lo}}_{\text{critic},\theta_1}$, $Q^{\text{lo}}_{\text{graph},\theta_2}$ and $\pi^{\text{lo}}_{\phi_1}$, high-level agent $Q^{\text{hi}}_{\theta_3}$ and $\pi^{\text{hi}}_{\phi_2}$
  if τ $\%$ graph update freq = 0 **then**
    Update Graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$ using FPS algorithm
  end if
end for

B Proofs of Theorems.

Derivation of equation 1. If a given policy π_{lo} requires n steps to get from current s to a goal g, the γ-discounted return is $Q_{\text{lo}}(s, \pi(s,g)|g) = (-1) + (-1)\gamma + (-1)\gamma^2 \cdots (-1)\gamma^{n-1} = -\frac{1-\gamma^n}{1-\gamma}$.

Thus, the temporal distance between s to $g(=n)$ is derived from $\gamma^n - 1 = (1-\gamma)Q_{\text{lo}}(s, \pi(s,g)|g)$ as

$$n = \log_\gamma \left(1 + (1-\gamma)Q_{\text{lo}}(s, \pi(s,g)|g)\right). \tag{3}$$

Definition B.1. $\mathcal{W}_{\mathbf{G}}(s_t, sg_t) = (wp_{t,0}, wp_{t,1}, ..., wp_{t,k})$ is a sequence of waypoint obtained by the graph search algorithm and $w(\mathcal{W}_{\mathbf{G}}, \tau) = wp_{t,i} \in \mathcal{W}_{\mathbf{G}}(s_t, sg_t)$ is the waypoint that is given to low-level policy at τ.

Given the transition distribution of the environment $\mathcal{T}(s_{\tau+1}|s_\tau, a_\tau)$, the transition data $(s_t, g_t, sg_t, r(s_{t+c_h}, g_t), s_{t+c_h})$ from the high-level policy's replay buffer has been obtained as

$$s_{t+c_h} = \prod_{\tau=t}^{t+c_h-1} \mathcal{T}(s_{\tau+1}|s_\tau, a_\tau) \cdot \pi_\beta^{\text{lo}}(a_\tau|s_\tau, w(\mathcal{W}_{\mathbf{G}_\beta}, \tau)), \tag{4}$$

where π_β^{lo} and $\mathbf{G}_\beta$ are the previous low-level policy and graph respectively. Also, by using a different graph $\mathbf{G}$ and an optimal policy $\pi^{\text{lo}*}$, we get a new transition data $(s_t, g_t, sg_t, r(s'_{t+c_h}, g_t), s'_{t+c_h})$, where

$$s'_{t+c_h} = \prod_{\tau=t}^{t+c_h-1} \mathcal{T}(s_{\tau+1}|s_\tau, a_\tau) \cdot \pi^{\text{lo}*}(a_\tau|s_\tau, w(\mathcal{W}_{\mathbf{G}}, \tau)). \tag{5}$$

For given s_t and s_{t+c_h}, we define the off-policy error rate, which is the normalized distance error with respect to the total traversal distance according to the change of π_β^{lo} and $\mathbf{G}_\beta$ to $\pi^{\text{lo}*}$ and $\mathbf{G}$, as

$$\rho(\mathbf{G}) = \frac{Dist(\psi(s'_{t+c_h}) \to \psi(s_{t+c_h}))}{Dist(\psi(s_t) \to \psi(s_{t+c_h}))}. \tag{6}$$

Lemma B.2. *Suppose that $Dist(\cdot \to \cdot)$ in Eq. (1) is Lipschitz continuous. Then, there exists a constant $L > 0$ such that $\forall x$ and y, $\max(Dist(x \to y), Dist(y \to x)) \leq L||x-y||$, where $||\cdot||$ is the Euclidean norm, since $Dist(x \to x) = 0$. Then, any $\epsilon/L-$resolution graph w.r.t the Euclidean norm, whose existence is trivial, is an $\epsilon-$resolution graph w.r.t $Dist(\cdot \to \cdot)$.*

Proof of Theorem 4.2

Proof. Let $\mathcal{C}^{s\to g}$ be one of the shortest paths from s to g and T be the distance of $\mathcal{C}^{s\to g}$. Also let $p_1 \in \mathcal{C}^{s\to g}$ be a point that $Dist(\psi(s) \to p_1) = c_l - \epsilon$. Then, $\exists wp_1 \in \mathbf{V}$ $s.t.$ $\max(Dist(p_1 \to wp_1), Dist(wp_1 \to p_1)) < \epsilon$, because $\mathbf{G}$ is an $\epsilon-$resolution graph. Since $Dist(\cdot \to \cdot)$ is a temporal distance, it satisfies the triangular inequality and then, $Dist(\psi(s) \to wp_1) \leq Dist(\psi(s) \to p_1) + Dist(p_1 \to wp_1) < (c_l - \epsilon) + \epsilon = c_l$ and $Dist(wp_1 \to g) \leq Dist(wp_1 \to p_1) + Dist(p_1 \to g) < \epsilon + (T - c_l + \epsilon) = T - c_l + 2\epsilon$.

Repeating the above procedure, let $p_{i+1} \in \mathcal{C}^{wp_i \to g}$ be a point that $Dist(wp_i \to p_{i+1}) = c_l - \epsilon$. Then, $\exists wp_{i+1} \in \mathbf{V}$ $s.t.$ $\max(Dist(p_{i+1} \to wp_{i+1}), Dist(wp_{i+1} \to p_{i+1})) < \epsilon$. Then, $Dist(wp_i \to wp_{i+1}) < c_l$ and $Dist(wp_{i+1} \to g) < T - (i+1)c_l + 2(i+1)\epsilon$. Consequently, the agent after T time-step will be closer than the $\lfloor T/c_l \rfloor^{th}$ waypoint from g. The remaining distance is less than

$$T - \lfloor T/c_l \rfloor c_l + 2\lfloor T/c_l \rfloor \epsilon. \tag{7}$$

Thus, if an agent follows the sequence of waypoints $\{s, wp_1, wp_2, ..., g\}$, which is generated from a graph search algorithm over $\mathbf{G}$ and $\pi^{\text{lo}*}$, the error rate over this path satisfies

$$\rho(\mathbf{G}) \leq \frac{T - \lfloor T/c_l \rfloor c_l + 2\lfloor T/c_l \rfloor \epsilon}{T} \leq \frac{T - (c_l - 2\epsilon)(T/c_l)}{T} = \frac{2\epsilon}{c_l}. \tag{8}$$

Thus the off-policy error rate ρ is equal or less than $2\epsilon/c_l$ during T. Since all path from s to g takes at least T time-steps, this upper-bound of error rate is also satisfied in all path from s to g. □

C Additional Results

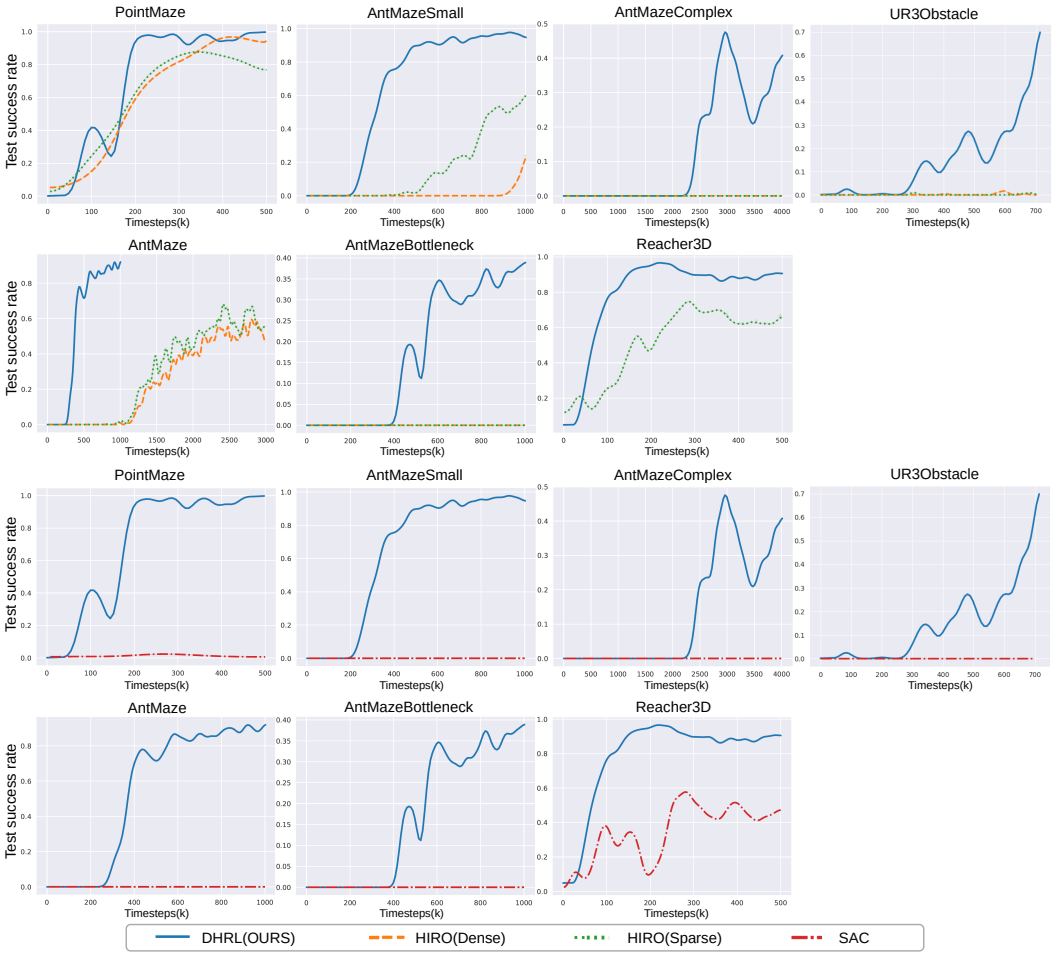

Figure 10: Comparison with shallow RL (SAC) and vanilla HRL (HIRO). The completely failed baselines are occluded by others.

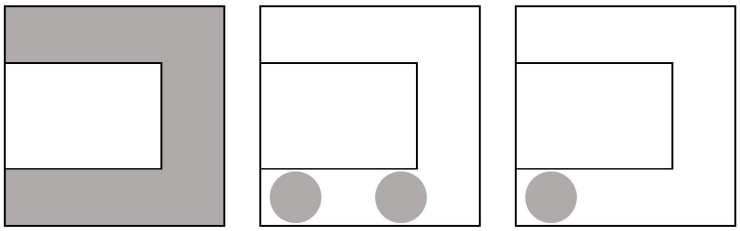

Figure 11: Examples of various initial state distributions.

Table 2: Performance of DHRL in various difficulties of initial state distributions.

SUCCESS RATE	EASY(UNIFORM)	MEDIUM(2 FIXED POINT)	HARD(1 FIXED POINT)
ANTMAZE 0.3M	80.4%	28.5%	12.2%
ANTMAZE 0.5M	87.1%	88.2%	71.5%

As shown in the table above, the wider the initial distribution, the easier it is for the agent to explore the map. In other words, the 'fixed initial state distribution' condition we experimented with in this paper is a more difficult condition than the 'uniform initial state distribution' that previous graph-guided RL algorithms utilize. Of course, 'fixed initial state distribution' requires less prior information about the state space. We further experimented with ours (DHRL) under various types of reset conditions as shown in Table 2. As expected, our algorithm shows faster exploration at the uniform reset point.

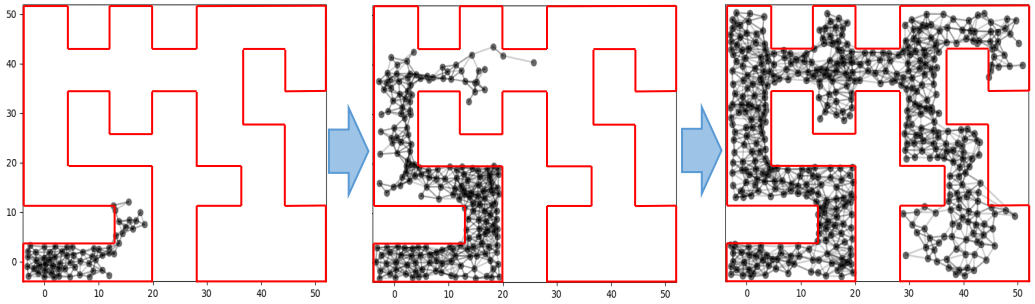

Figure 12: Changes in the graph level over the training; DHRL can explore long tasks with 'fixed initial state distribution' and limited knowledge about the environment.

Table 3: **Comparisons between our algorithm (DHRL) and baselines:** The numbers next to the environment names are the time-steps for training the models. The results are averaged over 4 random seeds and smoothed equally. '-D' and '-S' mean dense reward and sparse reward respectively. We use NVIDIA RTX A5000.

Success Rate		SAC	HIRO-D	HIRO-S	HRAC-D	HRAC-S	HIGL-D	HIGL-S	**DHRL**
PointMaze	0.25M	1.2%	73.8%	77.9%	6.1%	56.1%	25.5%	34.3%	**96.9%**
	0.5M	0.3%	93.6%	76.8%	33.5%	93.7%	88.3%	91.8%	**99.8%**
AntMazeSmall	0.5M	0.0%	0.0%	2.0%	9.2%	54.0%	36.3%	44.1%	**89.8%**
	1.0M	0.0%	24.1%	60.1%	83.4%	88.4%	83.7%	52.2%	**95.1%**
AntMaze	0.5M	0.0%	0.0%	0.0%	1.1%	0.0%	60.2%	32.7%	**71.5%**
	1.0M	0.0%	0.7%	0.8%	68.7%	48.9%	78.1%	60.3%	**91.1%**
Bottleneck	0.5M	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	**16.5%**
	1.0M	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	**38.7%**
Complex	2.5M	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	**20.4%**
	4.0M	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	**40.1%**
Reacher-3D	0.25M	49.2%	-	66.0%	-	26.5%	-	78.2%	**95.1%**
	0.5M	47.2%	-	67.1%	-	44.1%	-	47.1%	**90.6%**
UR3Obstacle		0.0%	2.5%	1.9%	0.5%	0.0%	11.1%	0.5%	**69.8%**

Table 4: Hyperparameters for HRL: When evaluating the previous HRL algorithms, we used the same hyperparameters as used in their papers. We also tried various numbers of landmarks and c_h which may affect the performance in long-horizon tasks.

	HIRO	HRAC	HIGL
high-level τ	0.005	0.005	0.005
π^{hi} lr	0.0001	0.0001	0.0001
Q^{hi} lr	0.001	0.001	0.001
high-level γ	0.99	0.99	0.99
high-level train freq	10	10	10
c_h		10-50	
low-level τ	0.005	0.005	0.005
π^{lo} lr	0.0001	0.0001	0.0001
Q^{lo} lr	0.001	0.001	0.001
low-level γ	0.95	0.95	0.95
hidden layer	(300,300)	(300,300)	(300,300)
number of coverage landmarks γ	-	-	20-250
number of novelty landmarks γ	-	-	20-250
batch size	128	128	128

Table 5: Hyperparameters for SAC

	SAC
hidden layer	(256, 256, 256)
actor lr	0.0003
critic lr	0.0003
entropy coef	0.2
τ	0.005
batch size	256
γ	0.99

Table 6: Hyperparameters for DHRL

	DHRL
hidden layer	(256, 256, 256)
initial episodes without graph planning	75
gradual penalty transition rate	0.2
high-level train freq	10
actor lr	0.0001
critic lr	0.001
τ	0.005
γ	0.99
number of landmarks	300-500
target update freq	10
actor update freq	2