Incorporating Spatial Information into Goal-Conditioned Hierarchical Reinforcement Learning via Graph Representations

Anonymous authors
Paper under double-blind review

Abstract

The integration of graphs with Goal-conditioned Hierarchical Reinforcement Learning (GCHRL) has recently gained attention, as intermediate goals (subgoals) can be effectively sampled from graphs that naturally represent the overall task structure in most RL tasks. However, existing approaches typically rely on domain-specific knowledge to construct these graphs, limiting their applicability to new tasks. Other graph-based approaches create graphs dynamically during exploration but struggle to fully utilize them, because they have problems passing the information in the graphs to newly visited states. Additionally, current GCHRL methods face challenges such as sample inefficiency and poor subgoal representation. This paper proposes a solution to these issues by developing a graph encoder-decoder to evaluate unseen states. Our proposed method, Graph-Guided sub-Goal representation Generation RL (G4RL), can be incorporated into any existing GCHRL method to enhance performance. We show that the graph encoder-decoder can be effectively implemented using a network trained on the state graph generated during exploration. Empirical results indicate that leveraging high and low-level intrinsic rewards from the graph encoder-decoder significantly enhances the performance of state-of-the-art GCHRL approaches with an extra small computational cost in dense and sparse reward environments.

1 Introduction

Traditional reinforcement learning methods face great challenges when learning policies in environments with long time horizons and sparse rewards. To address these challenges, Hierarchical Reinforcement Learning (HRL) methods have been proposed to break problems into smaller, more manageable scales suitable for efficient learning. Previous works (Sutton et al., 1999; Kulkarni et al., 2016; Vezhnevets et al., 2017; Levy et al., 2017) have shown its capability in large, sparse-reward environments. Among the HRL methods, Goal-Conditioned Hierarchical Reinforcement Learning (GCHRL) has attracted much attention because of its well-defined paradigm (Nachum et al., 2018b) and resemblance to the human thinking process. Although GCHRL has shown superior performance compared to non-hierarchical methods in certain scenarios, questions such as how to find better subgoal representations and explore the state space more efficiently (Nachum et al., 2018a; Guo et al., 2021) remain unanswered.

GCHRL methods typically utilize two levels of agents as stated in Nachum et al. (2018b). A high-level agent chooses the next target state based on its current state, while a low-level agent decides how to reach the target state specified by its high-level instructor. Both agents face their own challenges. The high-level agent suffers from sample inefficiency in environments with a large state-action space, a problem also encountered by non-hierarchical methods, while the low-level agent is trained solely by the reward signals derived from the distances between the current state and subgoals in the representation space, making the reward signals highly susceptible to bad subgoal representations.

Previous works have tried to either enhance the efficiency of high-level exploration (Huang et al., 2019; Zhang et al., 2022) or find a good subgoal space to boost GCHRL's performance in complex environments

(Wang et al., 2024). To the best of our knowledge, while these methods have made progress in their respective areas, no prior work has attempted to address them in combination. This integration is crucial, as uniting an effective high-level agent with an accurate low-level agent within a cohesive framework can yield performance improvements that surpass simple additive effects.

Combining graph theory with RL has recently become a trend in the community (Lee et al., 2022; Gieselmann & Pokorny, 2021) as graphs are inherently well-suited for representing the environment and task structure.

Previous work has focused on areas such as decision-making through graph search or traversal (Wan et al., 2021; Shang et al., 2019; Eysenbach et al., 2019), and as well as using graphs as world models (Zhang et al., 2021; Huang et al., 2019).

Yet, many previous works rely on pre-crafted graphs, which hinder the generalization of the methods. Additionally, most existing works use graphs constructed directly from the original state space, which cannot provide meaningful guidance when the agent encounters a state that is not represented in those graphs.

To utilize graph representations when a new state (node) is encountered, transitioning to these representations (Hamilton, 2020; Chen et al., 2020; Khoshraftar & An, 2024) through graph learning is a viable option. Recent studies have shown that employing various types of graph representations for learning can benefit the underlying RL problem (Klissarov & Precup, 2020; Klissarov & Machado, 2023).

In this paper, we propose a novel approach that simultaneously addresses all of the aforementioned problems. Specifically, we keep track of visited states, construct a state graph based on them until the number of nodes reaches a threshold, and then update the graph by adding and dropping nodes as new states are visited, all while building a subgoal space through graph learning based on state representation and spatial connection information provided by the graph. By generating subgoal representations through graph learning, these representations will reflect their relative positions in the decision chain, thereby creating a more effective subgoal space. In estimating the distance between the current state and the intended subgoal in the original space, we use the distance between their corresponding representations in the subgoal space. This distance is then used to calculate the intrinsic reward for curiosity-driven exploration, aiming to improve sample efficiency.

The main contributions of this paper are as follows:

- We propose a novel architecture that employs a graph encoder-decoder to summarize spatial information into subgoal representations, enabling the evaluation of newly visited states. This architecture can be integrated into any existing GCHRL algorithm to enhance performance.
- We present a method for online construction of the state graph as a graph world model (Ha & Schmidhuber, 2018; Zhang et al., 2021) for the agent by sampling from trajectories.
- We use novelty-based auxiliary rewards (Simsek & Barto, 2006; Nehmzow et al., 2013) derived from subgoal representations to improve sample efficiency for high-level and low-level agents.

We tested our approach on several environments (Todorov et al., 2012) to assess the significance of our experimental results. The findings indicate that our method can significantly enhance the performance of the underlying HRL approach in terms of both sample efficiency and success rate/cumulative reward.

2 Preliminaries

2.1 Notation

As the most common way to model reinforcement learning scenarios, we introduce the Markov Decision Process (MDP) (Puterman, 2014) defined as a tuple $< \mathcal{S}, \mathcal{A}, P, R, \gamma >$ as follows: At each time step t, the agent observes the current state $s_t \in \mathcal{S}$ provided by the environment and chooses an action $a_t \in \mathcal{A}$ based on its internal policy $\pi(a_t|s_t)$, which defines the probability of selecting an action given the current state. The action is then executed, and the interaction with the environment leads the agent to a new state

s_{t+1} according to a transition probability function $P(s_{t+1}|s_t, a_t)$ which is known only to the environment. Subsequently, the agent receives a reward r_t from the environment determined by a reward function $R(s_t, a_t)$ that evaluates the action taken in the current state and is also only visible to the environment. The agent aims to learn an optimal policy π to maximize the expected discounted cumulative reward $\mathbb{E}_{\pi,t:0\to T}\big[\sum_{i=0}^{T}\gamma^i r_i\big]$, where γ (with $0 \leq \gamma < 1$) is a pre-defined discount factor used to prioritize immediate rewards over distant future rewards, therefore preventing the total reward from becoming infinite.

2.2 Goal-conditioned Hierarchical RL (GCHRL)

Goal-Conditioned Reinforcement Learning (GCRL) trains agents to achieve specific goals, which are the target states. The agent receives an additional goal input along with the state input and learns a policy to achieve this goal. Goals are represented explicitly in the input to the policy, guiding the agent's actions towards achieving specific outcomes, and the reward function is often goal-dependent, providing positive feedback when the agent reaches the desired goal state.

To deal with large and complicated environments, Goal-Conditioned Hierarchical Reinforcement Learning (GCHRL) (Nachum et al., 2018b; Zhang et al., 2022; Wang et al., 2024) decomposes the learning task into a hierarchy of smaller, more manageable sub-tasks. Typically there are two levels of agents. At time step t, the high-level agent chooses a subgoal g_t which is the state representation of a target state that the high-level agent wants the low-level agent to achieve as a part of the overall task. This choice is made by sampling g_t from the high-level policy $\pi_h(g_t|\phi(s_t))$, where $\phi : s \mapsto \mathbb{R}^d$ is the state representation function which gives a condensed representation of the state.

Each state s_t can be mapped to its subgoal feature $g(s_t)$ by a subgoal feature extractor. Note that $g(s_t)$ is not the same as g_t. The former, $g(s_t)$, is the learned subgoal feature of the current state s_t, while the latter, g_t, is the target state we aim to reach from the state s_t in one step or multiple steps.

Given the subgoal g_t sampled from the high-level policy $\pi_h(g_t|\phi(s_t))$ for the current time step t and the state representation vector $\phi(s_t)$, a low-level agent executes action a_t based on the low-level policy $\pi_l(a_t|\phi(s_t), g_t)$. The low-level agent is trained using the intrinsic reward signal $r_{int}(s_t, g_t, a_t, s_{t+1}) = -\|\phi(s_{t+1}) - g_t\|^2$ to encourage it to achieve the subgoal.

Both agents can be implemented by any policy-based methods, including those introduced in previous works on policy gradients such as Fujimoto et al. (2018); Haarnoja et al. (2018) and Schulman et al. (2017).

2.3 Graph and MDP

Graph is a generic data structure, which can model complex relations among objects in many real-world problems. A graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, 2, \ldots, N\}$ is the set of nodes and $\mathcal{E} = \{e_{ij}\}$ is the set of edges without self-loops. The adjacency matrix of $\mathcal{G}$ is denoted by $\boldsymbol{A} = (\boldsymbol{A}_{i,j}) \in \mathbb{R}^{N\times N}$ with $\boldsymbol{A}_{i,j} = 1$ if there is an edge between nodes i and j, otherwise $\boldsymbol{A}_{i,j} = 0$. The adjacency matrix can be extended to a *weighted* adjacency matrix, where $\boldsymbol{A}_{i,j}$ is a weight of the edge e_{ij}.

In MDP, a node can represent a state, while the edge weights can model the transition probabilities or reachability statistics between states.

3 Methods

This section presents our framework, Graph-Guided subGoal representation Generation (G4RL). Our method reshapes the subgoal space utilizing a state graph to incorporate the relative spatial information of visited states.

One drawback of previous hierarchical reinforcement learning algorithms (Nachum et al., 2018b; Kim et al., 2021; Zhang et al., 2022; Luo et al., 2024) is the Euclidean distance calculated in the original state representation space between the current state and the intended goal does not reflect the true progress of the low-level agent, as there is rarely a straight path in the space from the current state to the subgoal. A low-level agent trained with such information may receive an inaccurate reward signal, thus impairing its

performance. Another issue is that the high-level agent may propose a subgoal that is too difficult to reach if its output is not constrained, wasting exploration steps on trying to reach those unreasonable subgoals (Zhang et al., 2022). Our proposed method aims to mitigate both problems by calculating the distance in a subgoal representation space between subgoal representations given by a graph encoder-decoder. This graph encoder-decoder takes into account the actual connectivity between states, ensuring that the generated subgoal representations respect adjacency information. This approach provides more accurate reward signals and leads to more reasonable target subgoals.

3.1 State graph

To record the visited states and their connections, we maintain a state graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a fixed number N of nodes[1]. This graph is built and updated during training, with no pre-training using expert data or handcrafted process involved in its construction.

Each node is labelled by the corresponding state and for each node s_t, the corresponding state representation vector $\phi(s_t)$, which is also referred to as the node feature, is stored. Edges in the graph represent connectivity between states. The graph is constantly updated during exploration.

3.1.1 Graph construction

The graph is initialized with N empty nodes and no edges. The corresponding weighted adjacency matrix $\boldsymbol{A}$ is set to an $N \times N$ zero matrix. We perform the GCHRL explore process using randomly initialized policy π_h and π_l. Once the agent encounters a state representation never seen before, that is, the representation vector is different from any state representations stored in the graph, as described in equation (1), it stores the state representation $\phi(s_t)$ as the node feature of an empty node in the graph and build an edge between this node and the node corresponds to the previous state.

$$\forall_{s_v \in \mathcal{V}}, \|\phi(s_t) - \phi(s_v)\|_2 > \epsilon_d, \tag{1}$$

$$\boldsymbol{A}_{s_t, s_{t-1}} = \boldsymbol{A}_{s_{t-1}, s_t} = 1, \tag{2}$$

where ϵ_d is a hyper-parameter controlling the distance threshold between state representations. When the agent encounters a state s_t with feature $\phi(s_t)$ that is similar to several node representations already stored in the graph, it finds the state whose representation is the closest to the current state feature:

$$s_v = \underset{s_u : \|\phi(s_t) - \phi(s_u)\|_2 \leq \epsilon_d}{\arg\min} \|\phi(s_t) - \phi(s_u)\|_2. \tag{3}$$

Then the node s_v is relabeled as s_t and the weight for the edge (s_{t-1}, s_t) is updated as follows:

$$\boldsymbol{A}_{s_{t-1}, s_t} = \boldsymbol{A}_{s_t, s_{t-1}} := \boldsymbol{A}_{s_{t-1}, s_t} + 1. \tag{4}$$

Note that a large weight indicates more frequent transitions between the underlying states.

We have used the Euclidean norm to define the distance between feature vectors. Since some elements may contain more spatial information than others, one can use a weighted Euclidean norm to define the distance between state representations instead.

3.1.2 Graph updating

The graph has a fixed number of nodes. Suppose the graph is now full. When a new state s_t is encountered, if s_v from equation (3) exists, as before we relabel the node as s_t and perform edge update as shown in equation (4); Otherwise, we replace the oldest state node in the graph with the current state node, delete all edges previously linked to that node, and create an edge (s_{t-1}, s_t) with weight $\mathbf{A}_{s_{t-1}, s_t} = \mathbf{A}_{s_t, s_{t-1}} = 1$. Alternatively, we could replace the state node most weakly connected to other nodes by finding the node with the least weighted sum of edges.

[1]The number of training states for the graph encoder-decoder grows quadratically with N because the adjacency weight matrix has N^2 elements. The choice of N depends on the machine's capabilities.

3.2 Graph encoder-decoder

To enable the assignment of suitable subgoal representations to every possible state, including unseen ones, we use node representations and edges to train a graph encoder-decoder. The parameter updates of the graph encoder-decoder and the policies during policy training are performed alternately in each episode. We do not train the graph encoder using expert policies then freeze its parameters for policy training.

The encoder-decoder starts training after the graph is full and continues periodically after processing a few trajectories. Section 3.3 will show the details of the training schedule.

The encoder $\mathbf{E}$ maps every state representation $\phi(s)$ to a subgoal representation $g(s)$. We use a feed-forward network (FFN) with several layers as the encoder $\mathbf{E}$:

$$g(s) = \mathbf{E}(\phi(s)) = \mathrm{FFN}(\phi(s)). \tag{5}$$

The weight parameters of the feed-forward network will be learned through training. The decoder $\mathbf{D}$ takes two subgoal representations as input and outputs the inner product of these two representations:

$$\mathbf{D}(g(s_u), g(s_v)) = g(s_u)^T g(s_v). \tag{6}$$

The aim is to use the encoder-decoder structure to predict node relations. Naturally we can use $\boldsymbol{A}_{s_u,s_v}$ as a measure of the relation between nodes s_u and s_v. But for the sake of numerical stability in the training process, we use $\boldsymbol{A}_{s_u,s_v} / \max_{s_i,s_j} \boldsymbol{A}_{s_i,s_i}$ as a measure. Thus the loss function is defined as:

$$\mathcal{L} = \sum_{s_u,s_v \in \mathcal{V}} \left[\mathbf{D}(g(s_u), g(s_v)) - \boldsymbol{A}_{s_u,s_v} / \max_{s_i,s_j} \mathbf{A}_{s_i,s_i}\right]^2. \tag{7}$$

This loss function can enforce the subgoal representation provided by the encoder to respect neighbouring features in the graph.

Note that in each training phase of the graph encoder-decoder (except the first one), we use the values of the parameters obtained from the last training phase as the initial point, which helps save computation cost.

3.3 Adaptive training schedule of the graph encoder-decoder

The data stored in the graph, which comprises state representations as node features and connection information in the weighted adjacency matrix $\boldsymbol{A}$, is constantly updated. Due to the nature of online training, the update rate varies across episodes. If we train the graph encoder-decoder at regular intervals, this may lead to high variance and data loss in earlier episodes, as well as over-training in later episodes. To avoid this, we propose an adaptive training schedule for the graph encoder-decoder in the following paragraph.

There are two types of data changing in the graph: edge update, as shown in equation (4), and node replacement. We introduce a variable c to track the weighted number of data changes. Since the replacement of nodes has a much higher impact on the data than the edge update, we add $N-1$ to c if a node replacement occurs, and add 1 to c if an edge update happens:

$$c = \begin{cases} c + N - 1, & \text{if a node replacement happens,} \\ c + 1, & \text{if an edge update happens.} \end{cases} \tag{8}$$

When this variable exceeds a certain value, specifically a tolerance β multiplied by the total number of non-diagonal elements $N^2 - N$ in the matrix $\boldsymbol{A}$. we perform one gradient update for the graph encoder-decoder, and then we reset c to 0.

3.4 Hierarchical agent with graph encoder-decoder

Our proposed method involves traditional goal-conditioned settings and a subgoal representation extractor implemented by a graph encoder-decoder.

The high-level policy $\pi_h(g_t|\phi(s_t))$ nominates a subgoal every K steps and is trained using environmental reward r_{ext}. The policy can be implemented by any policy-based RL algorithm that takes transition tuples $(s_t, g_t, a_t, r_t, s_{t+1}, g_{t+1})$ as input. To encourage it to propose a subgoal that is not too difficult to reach from the current state s_t for more efficient exploration, we add an intrinsic term to the high-level reward, considering the distance between the subgoal features of s_t and g_t in the subgoal space:

$$r_h(s_t, g_t, s_{t+1}) = r_{ext} + r_{int} = r_{ext} + \alpha_h \cdot \mathbf{D}(\mathbf{E}(\phi(s_t)), \mathbf{E}(g_t)), \tag{9}$$

where α_h is a hyperparameter that controls the significance of the intrinsic term in the high-level reward.

The low-level agent $\pi_l(a_t|\phi(s_t), g_t)$, however, operates in the subgoal space. While it still takes $\phi(s_t)$ and g_t as input and outputs an atomic action a_t, we compute the reward based on distances in the subgoal space:

$$r_l(s_t, g_t, a_t, s_{t+1}) = -\|\phi(s_{t+1}) - g_t\|^2 + \alpha_l \cdot \mathbf{D}(\mathbf{E}(\phi(s_{t+1})), \mathbf{E}(g_t)), \tag{10}$$

where α_l is a hyperparameter controlling the significance of the reward term in the low-level reward. By computing the intrinsic reward in the subgoal space rather than in the state space, the function provides high values when proposed subgoals are easy to reach from the current location and low values when subgoals are close in the original state space but difficult to reach from the current location. The low-level agent can also be any policy-based algorithm.

3.5 Balancing between speed and performance

Due to the excessive comparisons between the current state representation and the node features during graph updates, as well as the training cost of the graph encoder-decoder with a huge graph, our experiments show the GCHRL method, after incorporating our method, takes approximately twice as long as the original GCHRL method.

To reduce the extra cost, we need to either decrease the frequency of sampling candidates for node features, train the graph encoder-decoder with a subset of all available training data, or do both.

For the sampling frequency, instead of comparing the state representation with node features in each time step, we do it in every t_c time steps. This may significantly speed up our method while maintaining satisfactory performance.

To reduce the training data in each graph encoder-decoder training cycle, we randomly sample node pairs in the graph instead of using every node pair for training.

In the experiments section, we will present the training time and performance of G4RL with the above two techniques applied.

3.6 Algorithm: G4RL

Now we describe our proposed method in Algorithm 1.

4 Experiments

In this section, we present our experiments that combine G4RL with existing GCHRL methods to demonstrate its positive effects on enhancing their performances. We also show that the generated graph reasonably represents the overall structure of the task. Our experiments are conducted in a set of challenging environments.

4.1 Environment settings

We used AntMaze, AntGather, AntPush,and AntMaze-Sparse environments from the GYM MuJoCo library (Todorov et al., 2012). The first three involve manipulating a multi-armed robot to perform a set of real-world tasks, while AntMaze-Sparse is the sparse reward version of AntMaze, presenting a greater challenge

Algorithm 1 G4RL

Require: High-level policy $\pi_h(g|\phi(s))$, low-level policy $\pi_l(a|\phi(s), g)$, replay buffer $\mathcal{B}$, graph encoder $\mathbf{E}$, graph decoder $\mathbf{D}$, high-level action frequency K, significance hyperparameter α_h and α_l, tolerance hyperparameter β, number of episodes N, nunber of steps in one episode T.

```
n = 0
while n ≤ N do
    t = 0
    c = 0
    while t ≤ T do
        if t mod K = 0 then
            Execute the high-level policy π_h(g_t|φ(s_t)) to sample the subgoal g_t.
        else
            Keep the subgoal g_t unchanged.
        Execute the low-level policy π_l(a_t|φ(s_t), g_t) to sample the atomic action a_t.
        Sample reward r_t and next state s_{t+1}.
        Calculate r_h(s_t, g_t, s_{t+1}) and r_l(s_t, g_t, a_t, s_{t+1}) using (9) and (10).
        Collect experience (s_t, g_t, a_t, r_h, r_l, s_{t+1}) and update the replay buffer B.
        Update node representations and edge weights using collected experience.
        Update c using (8).
        if c ≥ β then
            Update graph encoder E with node representation and edge information in the graph.
            c = 0.
        t = t + 1
    Update low-level policy π_l(a|φ(s), g) using the chosen HRL algorithm.
    Update high-level policy π_h(g|φ(s)) using the chosen HRL algorithm.
    n = n + 1
```

as it only provides a reward upon reaching the goal. For the state representation dimension, we selected a subset of raw state dimensions that contains spatial information (e.g. coordinates and arm angles) to serve as the node representation for the graph encoder-decoder.

4.2 Experimental Comparisons

We incorporated G4RL in the following existing GCHRL methods:

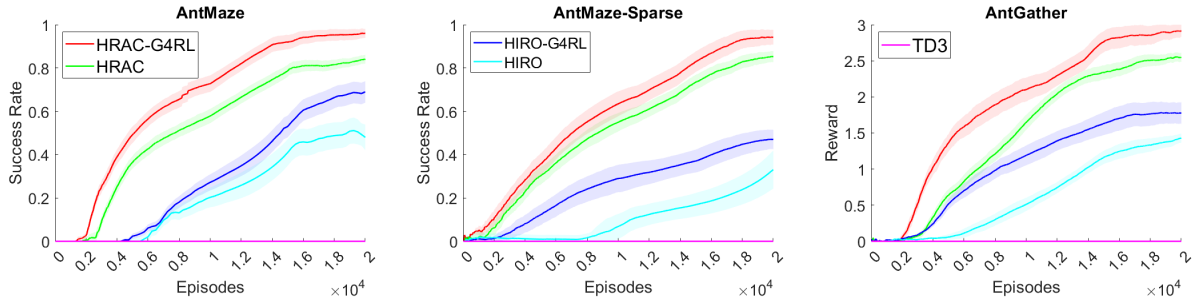
- **HIRO** (Nachum et al., 2018b): This is the first method which describes how the Goal-conditioned information can be integrated into hierarchical agents.
- **HRAC** (Zhang et al., 2022): This method enhances the performance of HIRO by training an adjacency network that produces subgoals easier to reach from the current subgoal.
- **HESS** (Li et al., 2022): This method applies a regularization term on consecutive subgoal representations in each update to stabilize the representation across episodes.
- **HLPS** (Wang et al., 2024): This method applies the Gaussian process on subgoal representations for a smoother representation update.

In addition to comparing these four GCHRL-G4RL methods with their native counterparts, we also compared them with the following non-hierarchical method:

- **TD3** (Fujimoto et al., 2018): This is a well-known non-hierarchical policy-based method designed for continuous action spaces and we use it to implement both high- and low-level agents.

Although reward is a key metric of an agent's learning ability, for AntMaze and AntMaze-Sparse, we compare success rates of these methods instead of their rewards on the corresponding tasks. This is because higher rewards in AntMaze/AntMaze-Sparse do not necessarily indicate better performance; the agent may achieve high rewards without reaching the goal.

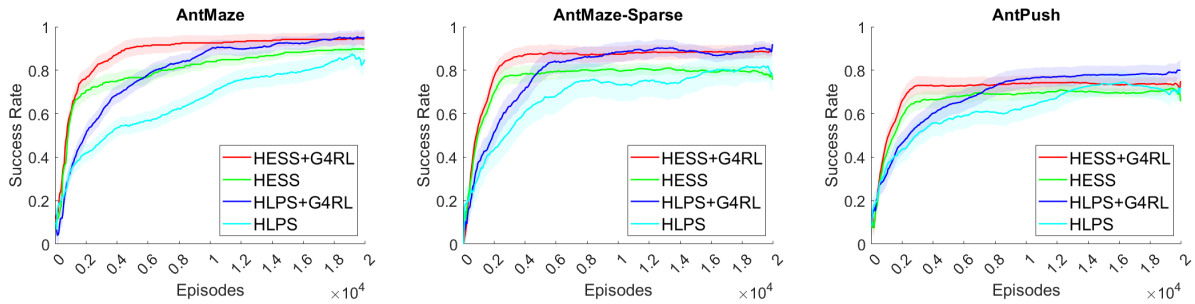
The learning curves of baseline methods and G4RL-applied versions are plotted in Figure 1. Note that all the curves reported in Section 4 are averages from 20 independent runs and they have been smoothed equally for better visualization.



(a) (b) (c)

Figure 1: Success Rate on (a) AntMaze (b) AntMaze-Sparse and Reward on (c) AntGather, using HIRO, HIRO-G4RL, HRAC, HRAC-G4RL, and TD3. The legends are distributed across sub-figures for better visualization.

From Figures 1 and 2, we observe that in all environments, incorporating G4RL in the base GCHRL method significantly enhances their performance, further improving the already strong results of these hierarchical methods compared to the non-hierarchical method.



(a) (b) (c)

Figure 2: Success Rate on (a) AntMaze (b) AntMaze-Sparse and (c) AntPush, using HESS, HESS-G4RL, HLPS, HLPS-G4RL.

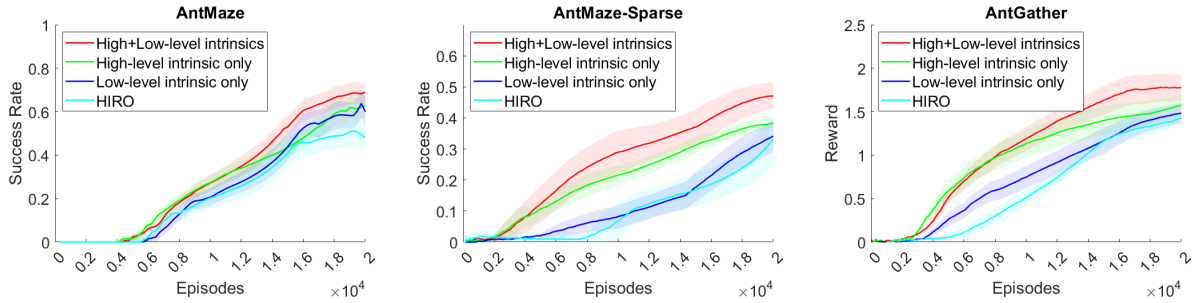
4.3 Ablation study

4.3.1 The effect of high/low-level intrinsic reward

This section discusses the different variants of G4RL to show the effectiveness of adding high-level and low-level intrinsic rewards. The following variants are compared:

- **High+Low-level intrinsics**: Apply both equation (9) and equation (10) to the high-level and low-level rewards respectively.
- **High-level intrinsic only**: Apply equation (9) to the high-level rewards and set $\alpha_l = 0$ in equation (10) when it is applied to the low-level rewards.
- **Low-level intrinsic only**: Apply equation (10) to the low-level rewards and set $\alpha_h = 0$ in equation (9) when it is applied to the high-level rewards.
- **HIRO/HRAC/HESS/HLPS**: Vanilla baseline methods.

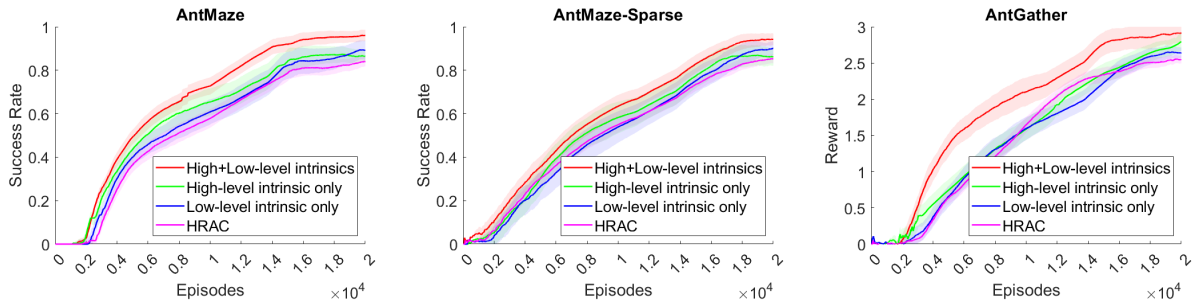
Same as before, all the curves reported in this section are drawn from results averaged across 20 independent runs. All curves have been smoothed equally for better visualization.



(a) (b) (c)

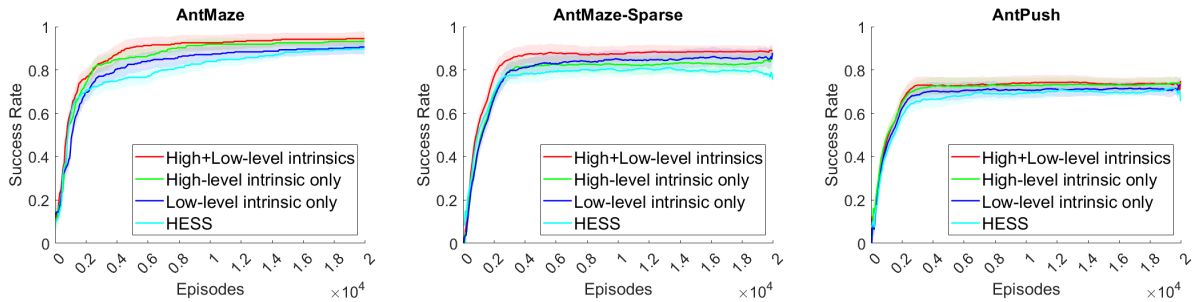
Figure 3: Success Rate on (a) AntMaze (b) AntMaze-Sparse and Reward on (c) AntGather using HIRO-G4RL, HIRO + High-level intrinsic, HIRO + Low-level intrinsic and HIRO.

Figures 3-6 show that for HIRO, HRAC, HESS and HLPS, G4RL with the intrinsic reward applied to both high-level and low-level agents achieves the best result across almost all of the tasks.



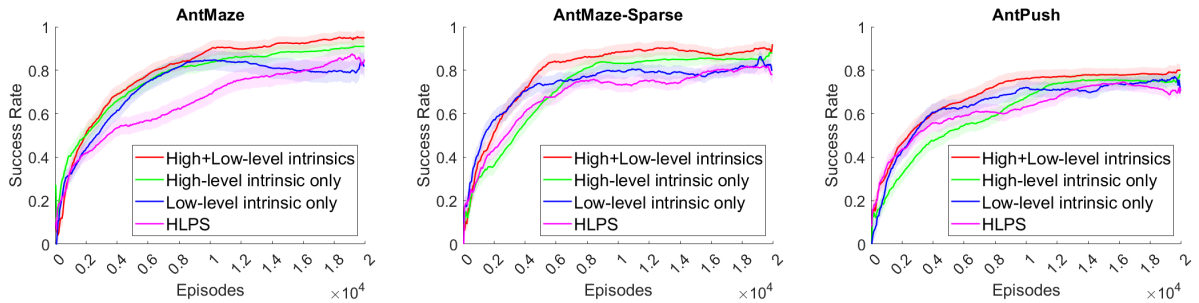
(a) (b) (c)

Figure 4: Success Rate on (a) AntMaze (b) AntMaze-Sparse and Reward on (c) AntGather using HRAC-G4RL, HRAC + High-level intrinsic, HRAC + Low-level intrinsic and HRAC.



(a) (b) (c)

Figure 5: Success Rate on (a) AntMaze (b) AntMaze-Sparse and (c) AntPush using HESS-G4RL, HESS + High-level intrinsic, HESS + Low-level intrinsic and HESS.



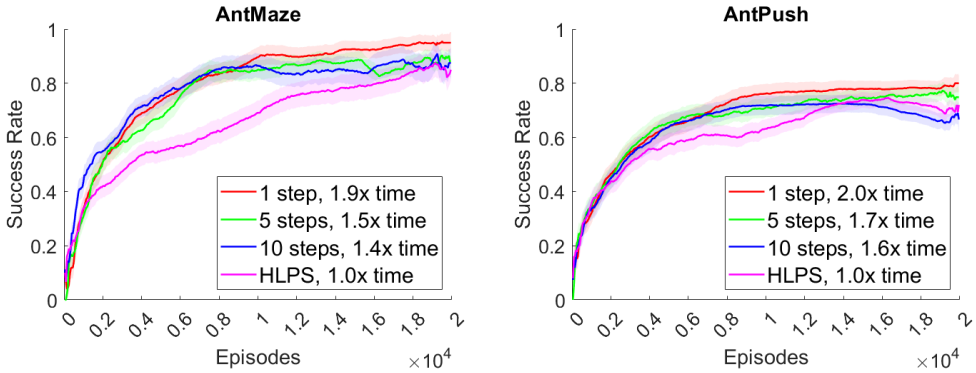
(a) (b) (c)

Figure 6: Success Rate on (a) AntMaze (b) AntMaze-Sparse and (c) AntPush using HLPS-G4RL, HLPS + High-level intrinsic, HLPS + Low-level intrinsic and HLPS.

Meanwhile, the high-level intrinsic-only variant performs better than the low-level intrinsic-only variant in our selected tasks. This suggests that in sparse reward tasks, the high-level intrinsic reward proposed in equation (9) can facilitate efficient exploration.

4.3.2 Balancing between time and performance

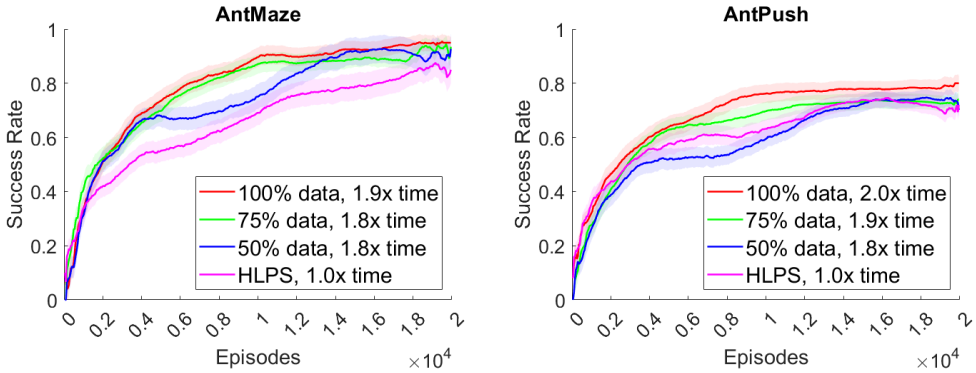
This section investigates the accelerating technique introduced in section 3.5. For the sampling frequency, we tested on sampling in each 1, 5, and 10 steps for HLPS.



(a) (b)

Figure 7: Success Rate on (a) AntMaze and (b) AntPush using HLPS+G4RL and HLPS. The number of steps in the legend indicates the selection of t_c as described in section 3.5 and the timescale is calculated w.r.t. the vanila HLPS algorithm.

For the percentage of training data used for the graph encoder-decoder, we tested on 50%, 75% and 100% for HLPS.



(a) (b)

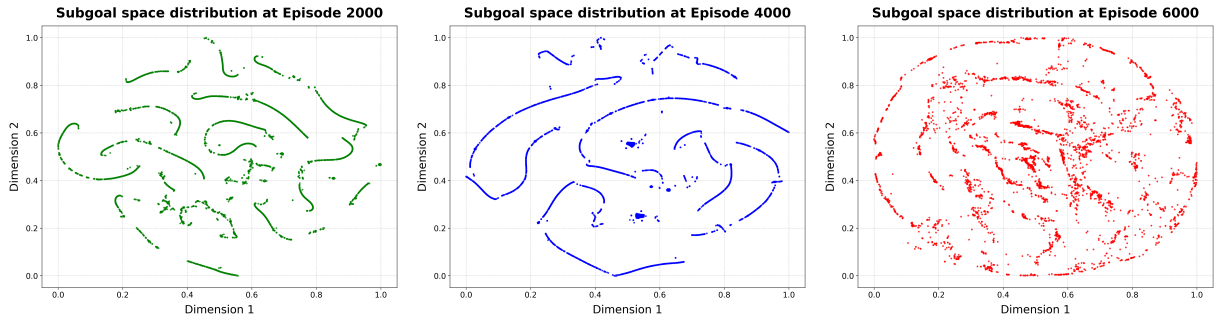
Figure 8: Success Rate on (a) AntMaze and (b) AntPush using HLPS+G4RL and HLPS. The percentage of data in the legend indicates the amount of data used in the training of the graph encoder-decoder as described in section 3.5 and the timescale is calculated w.r.t. the vanila HLPS algorithm.

Figures 7 and 8 show that reducing the sampling frequency significantly reduces our algorithm's time consumption with a minor performance loss. Reducing training data, however, doesn't contribute as much to the speedup.

4.4 Subgoal space visualization

This section shows how the subgoal space evolves in the AntMaze environment as the number of training episodes grows. We recorded state representations encountered in specific episodes and then used the corresponding graph encoders from those episodes to map these state representations to the subgoal representations. The subgoal representations are projected into 2D using PCA for visualization.

The distributions of subgoal representations in the subgoal space across different episodes are shown in Figure 9.



(a) (b) (c)

Figure 9: The distribution of subgoal representations in the subgoal space at the (a) 2000th (b) 4000th (c) 6000th episode.

From the figure we can conclude that the graph encoder-decoder gives better subgoal representations when the number of episodes grows. At the 2000th episode, the subgoal representations form a few chains in the 2D space, indicating that the current graph is partitioned into many isolated sub-graphs. The graph encoder has not yet learned to understand the environment well, and the agent cannot find relationships between different state representations yet. By the 4000th episode, the representations are less isolated, suggesting that the graph encoder has begun to learn the connections between different groups of state representation. Finally, at the 6000th episode, the subgoal representations are distributed more evenly across the entire space, indicating that the encoder can organize them into a highly connected graph. The subgoal space now better represents the underlying problem's structure.

The evolution of the graph encoder demonstrates that it can indeed learn to represent the task structure as the number of episodes grows.

5 Conclusion and future work

We present a novel approach using a graph encoder-decoder to mitigate the issues of poor subgoal representations and sample inefficiency in GCHRL. The architecture is designed to efficiently evaluate unseen states by operating in the graph representation space. It is easy to implement and can be integrated into any prevailing GCHRL algorithms for better performance. Additionally, we provide a comprehensive explanation of why the proposed architecture works, supported by experiments on both sparse and non-sparse control tasks that demonstrate the effectiveness and robustness of our method.

Generalizing knowledge learnt in one environment to other similar environments remains to be solved in reinforcement learning. In the future, we will extend our work by exploring how to generate subgoals with more interpretable representations for knowledge transfer with the help of other kinds of graph representations (e.g. Graph Laplacian). Another direction is to transfer the knowledge stored in the graph to other tasks by analyzing graph structures and mapping nodes of their state graphs.

References

Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. Graph representation learning: a survey. *APSIPA Transactions on Signal and Information Processing*, 9:e15, 2020.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pp. 1329–1338. PMLR, 2016.

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.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

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

Shangqi Guo, Qi Yan, Xin Su, Xiaolin Hu, and Feng Chen. State-temporal compression in reinforcement learning with the reward-restricted geodesic metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5572–5589, 2021.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

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.

William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

Zhiao Huang, Fangchen Liu, and Hao Su. Mapping state space using landmarks for universal goal reaching. *Advances in Neural Information Processing Systems*, 32, 2019.

Shima Khoshraftar and Aijun An. A survey on graph representation learning methods. *ACM Transactions on Intelligent Systems and Technology*, 15(1):1–55, 2024.

Junsu Kim, Younggyo Seo, and Jinwoo Shin. Landmark-guided subgoal generation in hierarchical reinforcement learning. *Advances in neural information processing systems*, 34:28336–28349, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Martin Klissarov and Marlos C Machado. Deep laplacian-based options for temporally-extended exploration. *arXiv preprint arXiv:2301.11181*, 2023.

Martin Klissarov and Doina Precup. Reward propagation using graph convolutional networks. *Advances in Neural Information Processing Systems*, 33:12895–12908, 2020.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Seungjae Lee, Jigang Kim, Inkyu Jang, and H Jin Kim. Dhrl: a graph-based approach for long-horizon and sparse hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 35: 13668–13678, 2022.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. *arXiv preprint arXiv:1712.00948*, 2017.

Siyuan Li, Jin Zhang, Jianhao Wang, Yang Yu, and Chongjie Zhang. Active hierarchical exploration with stable subgoal representation learning. *arXiv preprint arXiv:2105.14750*, 2022.

Yu Luo, Tianying Ji, Fuchun Sun, Huaping Liu, Jianwei Zhang, Mingxuan Jing, and Wenbing Huang. Goal-conditioned hierarchical reinforcement learning with high-level model approximation. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. *arXiv preprint arXiv:1810.01257*, 2018a.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018b.

Ulrich Nehmzow, Yiannis Gatsoulis, Emmett Kerr, Joan Condell, Nazmul Siddique, and T Martin McGuinnity. Novelty detection as an intrinsic motivation for cumulative learning robots. *Intrinsically Motivated Learning in Natural and Artificial Systems*, pp. 185–207, 2013.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Wenling Shang, Alex Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Learning world graphs to accelerate hierarchical reinforcement learning. *arXiv preprint arXiv:1907.00664*, 2019.

Ozgur Simsek and Andrew G Barto. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the 23rd international conference on Machine learning*, pp. 833–840, 2006.

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

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

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*, pp. 3540–3549. PMLR, 2017.

Guojia Wan, Shirui Pan, Chen Gong, Chuan Zhou, and Gholamreza Haffari. Reasoning like human: Hierarchical reinforcement learning for knowledge graph reasoning. In *International Joint Conference on Artificial Intelligence*. International Joint Conference on Artificial Intelligence, 2021.

Vivienne Huiling Wang, Tinghuai Wang, Wenyan Yang, Joni-Kristian Kämäräinen, and Joni Pajarinen. Probabilistic subgoal representations for hierarchical reinforcement learning. *arXiv preprint arXiv:2406.16707*, 2024.

Lunjun Zhang, Ge Yang, and Bradly C Stadie. World model as a graph: Learning latent landmarks for planning. In *International conference on machine learning*, pp. 12611–12620. PMLR, 2021.

Tianren Zhang, Shangqi Guo, Tian Tan, Xiaolin Hu, and Feng Chen. Adjacency constraint for efficient hierarchical reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45 (4):4152–4166, 2022.

A Implementation details

A.1 Environment details

AntMaze This environment is a part of the Gymnasium-Robotics libraries. The size of the environment is 24×24. Both of the state space and the action space are continuous, with a state dimension of 31 and an action dimension of 8. The reward of each step is defined by its negative Euclidean distance from the current location to the target position. At evaluation time, the goal is set to $(0, 16)$ and an episode is recognized as successful if the agent is within an Euclidean distance of 5 from the goal.

AntMaze Sparse This environment is a variant of Antmaze. The size of the environment is 20×20. The state and action spaces are the same as those in AntMaze. The reward of each step is 1 if and only if the agent reaches within an Euclidean distance of 1 from the goal, which is set to $(2, 9)$.

AntGather This environment is described in Duan et al. (2016). The size of the environment is 20×20. The state and action spaces are continuous. The task involves gathering apples to the designated place. The

agent will be awarded $+1$ for each apple gathered and -1 for each bomb gathered. Apples and bombs are randomly placed in the 20×20 world.

AntPush The size of the environment is 20×20. The state and action spaces are continuous. A challenging task that requires both task and motion planning. The agent needs to move to the left then move up and push the block to the right in order to reach the goal.

A.2 Network architecture details

Our network architecture for the HRL agents is the same as described in Nachum et al. (2018b), Zhang et al. (2022), Li et al. (2022) and Wang et al. (2024). For HIRO, HRAC, HESS and HLPS both the high-level and low-level agents use the TD3 algorithm. The size of each hidden layer in actor and critic networks in TD3 is 300.

For the graph encoder-decoder, we use a four-layer fully connected network with the hidden size of 128. The activation function used is ReLU. The decoder is a dot product of the two input subgoal representations.

We use Adam as the optimizer for the actor network, critic network and the graph encoder is Adam optimizer (Kingma & Ba, 2014).

B Hyperparameters

In this section we list all hyperparameters used in our experiments.

Hyperparameters	**Values**
High-level agent	
Actor learning rate	0.0001
Critic learning rate	0.001
Batch size	128
Discount factor γ	0.99
Policy update frequency	1
High-level action frequency	10
Replay buffer size	20000
Exploration strategy	Gaussian($\sigma = 1$)
Low-level agent	
Actor learning rate	0.0001
Critic learning rate	0.001
Batch size	128
Discount factor γ	0.99
Policy update frequency	1
Replay buffer size	20000
Exploration strategy	Gaussian($\sigma = 1$)

Table 1: Hyperparameters used in high- and low-level TD3 agents.

Hyperparameters	**Values**
Number of nodes N	200
Batch size	128
Optimizer learning rate	0.0001
ϵ_d	0.1 for AntMaze/0.2 for others
α_h	0.1
α_l	0.1
β	0.2

Table 2: Hyperparameters used in the graph encoder-decoder.