From Connectivity to Rewards: Dense Reward Learning with Directed State Graphs

Anonymous authors
Paper under double-blind review

Abstract

The integration of graphs with Goal-Conditioned Hierarchical Reinforcement Learning (GCHRL) has received increasing attention, as graphs naturally encode task hierarchies for effective subgoal sampling. However, existing methods often overlook intrinsic connectivity information, failing to fully leverage the underlying topology for efficient learning. Most graph-based GCHRL methods use the graph as a stochastic sampling tool rather than as an environmental model that encodes connectivity and state-accessibility information. This limitation is particularly acute in quasimetric environments, where the inherent asymmetry of state transitions poses a fundamental challenge to stable policy learning and robust path planning. In this paper, we address these problems by introducing a state connectivity model designed to predict pairwise state connectivity strength in asymmetric environments. We transform these connectivity strengths into scalar auxiliary dense rewards, providing continuous guidance across multiple hierarchical levels. We demonstrate that our proposed framework, Graph-Guided Quasimetric Dense Reward (G2QDR), can be integrated into any existing GCHRL architecture to boost performance, and the state connectivity model is efficiently implemented via a neural network trained on a directed state graph generated during exploration. Empirical results across a wide range of sparse reward environments indicate that G2QDR significantly enhances the performance of baseline GCHRL approaches with minimal computational overhead.

1 Introduction

Sparse rewards remain a fundamental challenge in reinforcement learning (RL), where agents must learn from infrequent and delayed feedback signals. In such settings, traditional, non-hierarchical approaches (Schulman et al., 2017; Fujimoto et al., 2018; Haarnoja et al., 2018) often struggle with inefficient exploration and poor credit assignment. Goal-Conditioned Hierarchical Reinforcement Learning (GCHRL) addresses this issue by decomposing a long-horizon task into a sequence of intermediate subgoals: a high-level policy proposes subgoals, while a low-level policy learns to achieve them. This hierarchical decomposition significantly improves learning efficiency by transforming a difficult global objective into a set of more tractable local problems.

Despite these advantages, early GCHRL methods (Nachum et al., 2018a;b) often suffer from sample inefficiency, as the high-level agent must select subgoals from the entire state space—a task that can be even more challenging than selecting actions in non-hierarchical approaches. Prior work has attempted to address this issue by constraining subgoal selection to local regions (Zhang et al., 2022), imposing smoothness constraints on sampled subgoals (Li et al., 2022), or employing stochastic sampling strategies (Wang et al., 2024; 2025). While these approaches reduce the burden on the high-level agent, they do not organize visited states or subgoals into a structured representation, thereby losing important connectivity information.

In contrast, graph-based GCHRL methods (Lee et al., 2022; Gieselmann & Pokorny, 2021) model relationships and connectivity between states explicitly through a state graph, which is inherently well-suited for representing the environment and task structure. Previous graph-based work has explored directions such as decision-making via graph search or traversal (Wan et al., 2021; Shang et al., 2019; Eysenbach et al., 2019),

as well as the use of graphs as world models (Zhang et al., 2021; Huang et al., 2019). However, many of these approaches rely on pre-crafted graphs, which limit their generalizability. Moreover, most existing methods (Zhu et al., 2022; Hong et al., 2022) operate directly on the constructed graph, making them less effective when the agent encounters states that are not represented in the graph.

Graph learning, alternatively, enables the agent to leverage relational structure even when the current state is not explicitly represented in the graph, by generalizing through learned embeddings and connectivity patterns. Recent studies (Zhang et al., 2025) have shown that incorporating graph representations into the learning process can significantly enhance the performance of underlying reinforcement learning algorithms, improving both sample efficiency and generalization. Moreover, graph learning (Klissarov & Precup, 2020; Klissarov & Machado, 2023) facilitates more informed exploration and planning by capturing the topology of the state space, allowing agents to reason about long-range dependencies and transitions beyond their immediate experience.

Another common challenge in graph-based methods is that distances between states are often quasimetric (Wang & Isola, 2022; Wang et al., 2023), meaning that transition costs are asymmetric: moving from B to A can be significantly more difficult than moving from A to B. Undirected state graphs Zhang et al. (2025) fail to capture this asymmetry, motivating the use of directed graphs instead.

In this paper, we propose a graph-based online GCHRL framework that addresses these challenges. We construct a directed state graph online during exploration, incrementally incorporating visited states while pruning outdated nodes and connections. We then train a model on this graph to predict state connectivity, which serves as a proxy for transition distance. This connectivity measure is subsequently used to generate dense reward signals, enabling more efficient learning and improved understanding of the environment.

The main contributions of this paper are as follows:

- We introduce a method for the online construction of a directed state graph as a graph-based environmental model (Ha & Schmidhuber, 2018; Zhang et al., 2021). The graph is built directly during exploration, eliminating the need for expert data or manually designed structures.
- We propose a novel architecture that incorporates a state connectivity model to predict true state transitions based on observed arrivals. This model allows the evaluation of newly visited states that are not yet represented in the graph.
- We leverage auxiliary rewards (Simsek & Barto, 2006; Nehmzow et al., 2013), derived from state connectivity, to improve sample efficiency for the high-level agent and to provide well-calibrated learning signals for the low-level agent.
- Our architecture is compatible with any GCHRL algorithm in both symmetric and asymmetric environments, leading to improved performance across a broad range of tasks.

We evaluated our approach across a range of MuJoCo environments (Todorov et al., 2012) to assess the significance of our experimental results. The results show that our method improves the performance of the underlying GCHRL framework in terms of both sample efficiency and success rate.

2 Preliminaries

2.1 Markov Decision Processes

As the most widely used framework for modeling reinforcement learning problems, the Markov Decision Process (MDP) (Puterman, 2014) is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$. At each time step t, the agent observes the current state $s_t \in \mathcal{S}$ provided by the environment and selects an action $a_t \in \mathcal{A}$ according to its policy $\pi(a_t \mid s_t)$, which specifies the probability of choosing action a_t given state s_t.

Once the action is executed, the environment transitions the agent to a new state s_{t+1} according to the transition probability function $P(s_{t+1} \mid s_t, a_t)$, which is typically unknown in model-free settings. The agent

then receives a reward r_t determined by the reward function $R(s_t, a_t)$, which evaluates the action taken in the current state and is also not directly accessible to the agent.

The objective of the agent is to learn an optimal policy π that maximizes the expected discounted cumulative reward $\mathbb{E}_\pi\big[\sum_{t=0}^{T} \gamma^t r_t\big]$, where $\gamma \in [0, 1)$ is a predefined discount factor that prioritizes immediate rewards over those in the distant future, ensuring the total return remains finite.

2.2 Goal-conditioned Hierarchical RL (GCHRL)

Goal-Conditioned Reinforcement Learning (GCRL) (Liu et al., 2023) extends standard reinforcement learning by training agents to achieve specific goals, typically defined as target states. In this setting, the agent receives an additional goal input g_t alongside the state s_t and learns a goal-conditioned policy $\pi(a_t \mid s_t, g_t)$ that aims to reach the desired goal. By explicitly incorporating goals into the policy input, the agent's behavior is guided toward achieving specified outcomes. The reward function is often goal-dependent, providing positive feedback when the agent successfully reaches the target state.

To address the challenges posed by large and complex environments, Goal-Conditioned Hierarchical Reinforcement Learning (GCHRL) (Nachum et al., 2018b; Zhang et al., 2022; Wang et al., 2024) decomposes the task into a hierarchy of simpler and more manageable sub-tasks. Typically, this framework consists of two levels of control. For every p steps, the high-level agent selects a subgoal g_t, representing an intermediate target state, which is then passed to a low-level agent for execution. The subgoal is sampled from a high-level policy $\pi_h(g_t \mid \phi(s_t))$, where $\phi : \mathcal{S} \to \mathbb{R}^d$ denotes a state representation function that maps the raw state to a compact feature space.

Given the subgoal g_t and the state representation $\phi(s_t)$, the low-level agent selects an action a_t according to the policy $\pi_l(a_t \mid \phi(s_t), g_t)$. The low-level policy is trained using an intrinsic reward signal defined as $r_{\text{int}}(s_t, g_t, a_t, s_{t+1}) = -\|\phi(s_{t+1}) - g_t\|_2^2$ which encourages the agent to minimize the distance between the achieved state representation and the subgoal.

Both the high-level and low-level agents can be implemented using policy-based reinforcement learning methods, including those developed in prior work on policy gradients, such as Fujimoto et al. (2018); Haarnoja et al. (2018); Schulman et al. (2017).

2.3 Graph abstraction of MDP

A graph is a general and flexible data structure for modeling complex relationships among objects in many real-world problems. A graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $|\mathcal{V}| = N$ denotes the set of nodes and $\mathcal{E} = \{e_{ij}\}$ denotes the set of edges (without self-loops). The adjacency matrix of $\mathcal{G}$ is given by $\boldsymbol{A} = (\boldsymbol{A}_{i,j}) \in \mathbb{R}^{N \times N}$, where $\boldsymbol{A}_{i,j} = 1$ if there exists an edge between nodes i and j, and $\boldsymbol{A}_{i,j} = 0$ otherwise.

This formulation can be naturally extended to a weighted adjacency matrix, in which $\boldsymbol{A}_{i,j}$ represents the weight associated with edge e_{ij}, capturing the strength or significance of the relationship.

In the context of Markov Decision Processes (MDPs), nodes can be interpreted as states, while edge weights encode transition probabilities or reachability statistics between states. Under this perspective, the graph serves as an abstract representation of the environment dynamics, capturing how states are interconnected through agent–environment interactions (Lee et al., 2022).

More generally, such a graph can be viewed as a compressed or learned structural model of the MDP Zhang et al. (2025), where the connectivity reflects feasible transitions and the edge weights quantify their likelihood or frequency. This abstraction is particularly useful in large or continuous state spaces, where explicitly modeling the full transition function is intractable. By operating on the graph structure, one can exploit relational information between states to facilitate planning, exploration, and representation learning.

3 Methods

Previous work Zhang et al. (2025) focuses on deriving dense rewards using undirected graphs, but this approach encounters difficulties in quasimetric environments, where symmetry between states does not hold. In this section, we present our framework, Graph-Guided Quasimetric Dense Reward (G2QDR), which explicitly models the state connectivity with a directed graph. By integrating auxiliary rewards derived from transition information with the original sparse rewards for both high- and low-level agents, the policy learning process more rapidly adapts to the underlying state connectivity structure. This improves sample efficiency by encouraging more plausible subgoal proposals and better leveraging implicit spatial information.

3.1 State graph

To record the visited states and their relationships, we maintain a directed state graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a fixed number N of nodes[1]. This graph is built incrementally during the exploration phase of training, without relying on expert data or a handcrafted process.

Each node is associated with a state. For a node corresponding state s_t, we store the corresponding state representation vector $\phi(s_t) \in \mathbb{R}^d$ as the node feature and use s_t as the node label for simplicity. The edges in the graph then represent the connectivity between states.

We model the state graph as directed to capture the potentially asymmetric nature of transitions in quasimetric environments; directed edges naturally encode these directional discrepancies.

3.1.1 Incremental 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 explore the environment using the backbone agent with randomly initialized high- and low-level policies π_h and π_l. Whenever the agent encounters a previously unseen state representation—i.e., one that is sufficiently distant from all state representations currently stored in the graph, as defined in Equation (1), the state is added to the graph as a node, with the representation $\phi(s_t)$ as its feature. We define a window of size W, such that the W most recent preceding states are considered connected to the current state. Accordingly, we create edges between the current node and the nodes corresponding to these preceding states, as shown in equation (2).

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

$$\boldsymbol{A}_{s_{t-w}, s_t} = w^{-p}, \quad w = 1, \ldots, W, \tag{2}$$

where ϵ_d is a hyperparameter that specifies the distance threshold between state representations, and $p > 0$ is a hyperparameter that controls the polynomial decay's exponent. Naturally, we initialize the edge weight using the decay w^{-p}, as states that are farther apart in the trajectory (i.e., larger w) are considered more weakly connected.

When the agent encounters a state s_t with representation $\phi(s_t)$ that is similar to some node features 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, we treat the state s_t as s_v, relabel the node s_v as s_t, and relabel the corresponding edges accordingly. The weight for the edge (s_{t-w}, s_t) is then updated as follows:

$$\boldsymbol{A}_{s_{t-w}, s_t} := \boldsymbol{A}_{s_{t-w}, s_t} + w^{-p}, \quad w = 1, \ldots, W. \tag{4}$$

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

[1]The number of training examples for the state connectivity model grows quadratically with N because the adjacency weight matrix has N^2 elements. The choice of N depends on the machine's capabilities.

We use the Euclidean norm to measure the distance between representation vectors. However, since some components may encode more spatial information than others, a weighted Euclidean norm can alternatively be employed to define distances between state representations, allowing the metric to better reflect the structure of the environment.

3.1.2 Updating in a full graph

The graph has a fixed number of nodes. Suppose the graph is full, when a new state s_t is encountered, if s_v from equation (3) exists, as before we relabel the node as s_v 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 initialize the edges using equation (2). Alternatively, one could replace the node that is most weakly connected to the rest of the graph—that is, the node with the lowest sum of edge weights.

3.2 State connectivity model

To assign appropriate connectivity strengths to all possible state pairs, including those not yet observed, we use node features and edges to train a state connectivity model, which predicts connectivity strengths between unseen states whose representations are not stored in the graph.

The model $\mathcal{C}_\theta$ maps every paired state representation $\psi(s_u, s_v)$, which is a column vector, to a connectivity score c, where θ denotes its parameters. We instantiate $\mathcal{C}_\theta$ as a multi-layer feed-forward network (FFN):

$$c = \mathcal{C}_\theta(\psi(s_u, s_v)) = \mathrm{FFN}(\psi(s_u, s_v)). \tag{5}$$

The state connectivity model and the policies are updated alternately during policy training.

Order-sensitive paired state representation Since transition distances between states are often asymmetric, the paired-state representation should be order-dependent to enable learning quasimetric state connectivities (i.e. $\mathcal{C}_\theta(\psi(s_u, s_v)) \neq \mathcal{C}_\theta(\psi(s_v, s_u))$). Here, we propose some candidates for order-sensitive paired state representation:

- **Vector Concatenation** The input $\psi(s_u, s_v) = \begin{bmatrix}\phi(s_u)\\ \phi(s_v)\end{bmatrix} \in \mathbb{R}^{2d}$ is just a simple concatenation of the two state representation vectors. In this case, the intermediate layers of $\mathcal{C}_\theta$ learn higher-level features of the paired input $\psi(s_u, s_v)$.
- **Gated Fusion** The input $\psi(s_u, s_v) \in \mathbb{R}^d$ is computed via gated fusion parameterized by learnable weights $\boldsymbol{W} \in \mathbb{R}^{d\times 2d}$ and biases $\boldsymbol{b} \in \mathbb{R}^d$:

$$g(s_u, s_v) = \sigma(\boldsymbol{W}\begin{bmatrix}\phi(s_u)\\ \phi(s_v)\end{bmatrix} + \boldsymbol{b}) \in \mathbb{R}^d,\ \psi(s_u, s_v) = g(s_u, s_v) \odot \phi(s_u) + (1 - g(s_u, s_v)) \odot \phi(s_v), \tag{6}$$

  where σ is a gating function. Gated fusion learns the contribution of each input at the feature level. Instead of enforcing a fixed combination, it enables data-dependent weighting between the two inputs. Consequently, the representations $\psi(s_u, s_v)$ and $\psi(s_v, s_u)$ are distinctly different, rather than being related by a simple permutation.

Training loss Naturally we can use $\boldsymbol{A}_{s_u,s_v}$ as a measure of connectivity between nodes s_u and s_v. However, to constrain the scale of the network outputs during training, we instead normalize this quantity as $\boldsymbol{A}_{s_u,s_v} / \max_{s_i,s_j} \boldsymbol{A}_{s_i,s_j}$.

Accordingly, the loss function is defined as:

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

This loss function encourages the model's predictions to preserve local neighbourhood structure in the graph.

Note that in each training phase of the model (except the first), the parameters are initialized using the values learned in the previous phase, which serves as a good initial point.

3.3 Induced dense reward

Our proposed method introduces induced dense reward signals across all levels of agents in traditional goal-conditioned settings, enabling better utilization of experienced environmental dynamics and promoting more efficient learning.

High-level constraint reward The high-level policy $\pi_h(g_t|\phi(s_t))$ proposes a subgoal every P steps and is trained using the external environmental reward r_{ext}. It can be implemented using any policy-based reinforcement learning algorithm that operates on transition tuples (s_t, g_t, r_t, s_{t+1})

To promote more efficient exploration, we encourage the policy to generate subgoals that are easier to reach from the current state s_t. Specifically, we augment the high-level reward with an intrinsic term that rewards strong connections between the current state s_t and generated subgoal g_t. The resulting reward is defined as follows:

$$r_h(s_t, g_t, s_{t+1}) = r_{\text{ext}} + r_{\text{int}} = r_{\text{ext}} + \alpha_h \cdot \mathcal{C}_\theta(\psi(s_t, g_t)), \tag{8}$$

where $\alpha_h \geq 0$ is a hyperparameter that controls the significance of the intrinsic term in the high-level reward.

Low-level correction reward The low-level policy $\pi_l(a_t|\phi(s_t), g_t)$ can be implemented using any policy-based reinforcement learning algorithm that operates on transition tuples $(s_t, g_t, a_t, r_t, s_{t+1})$. In the traditional low-level policy learning process (Nachum et al., 2018b), the policy learns exclusively from reward signals derived from the Euclidean distance between the subsequent state representation and the subgoal. However, this approach is limited as it ignores environmental connectivity and fails to reflect the (true) transition distance between states accurately. We address this limitation by introducing a connectivity-based term as follows:

$$r_l(s_t, g_t, a_t, s_{t+1}) = -\|\phi(s_{t+1}) - g_t\|^2 + \alpha_l \cdot \mathcal{C}_\theta(\psi(s_{t+1}, g_t)), \tag{9}$$

where $\alpha_l \geq 0$ is a hyperparameter controlling the significance of the reward term in the low-level reward.

Asymmetric penalty In certain quasimetric environments, asymmetric transitions can prevent an agent from reaching its goal (e.g., one-way dead ends, pits, or cliffs). In such cases, a state s_v may be easily reachable from s_u, while the reverse transition from s_v to s_u is significantly more difficult or unlikely. To capture this asymmetry, we compare the learned directed connectivity scores $\mathcal{C}_\theta(\psi(s_u, s_v))$ and $\mathcal{C}_\theta(\psi(s_v, s_u))$, which serve as proxies for transition ease in each direction.

We define penalty terms for both the high- and low-level policies as:

$$r_{hp} = -\alpha_{hp} \cdot \max(\mathcal{C}_\theta(\psi(s_t, g_t)) - \mathcal{C}_\theta(\psi(g_t, s_t)), 0), \quad r_{lp} = -\alpha_{lp} \cdot \max(\mathcal{C}_\theta(\psi(s_t, s_{t+1})) - \mathcal{C}_\theta(\psi(s_{t+1}, s_t)), 0). \tag{10}$$

where $\alpha_{hp} \geq 0$ and $\alpha_{lp} \geq 0$ are hyperparameters, which control the strength of the penalty. The asymmetric gap $\mathcal{C}_\theta(\psi(s_u, s_v)) - \mathcal{C}_\theta(\psi(s_v, s_u))$ quantifies directional irreversibility. A large positive gap indicates that the transition from s_u to s_v is substantially easier than the reverse, suggesting a potentially hazardous or trapping structure in the environment.

The use of the $\max(\cdot, 0)$ operator ensures that only risky asymmetry is penalized. Without this clipping, the term could encourage the agent to favour transitions that are difficult to reach but easy to leave, which is an undesirable and unintended behavior. This penalty encourages both high- and low-level policies to remain within regions of the state space that exhibit more balanced bidirectional connectivity, corresponding to more reversible and safer transitions.

3.4 Evolution of dense signal strength

Due to the varying quality of the dense signal across different training stages,we introduce a stage-dependent controller λ to modulate the induced reward terms. Specifically, the hyperparameters α_h, α_l, α_{hp}, and α_{lp}

are all scaled by λ. Let n_t denote the current episode, with the activation phase spanning episodes n_1 to n_2 and the annealing phase spanning episodes n_3 to n_4. The controller is defined as:

$$\lambda = \begin{cases} 0 & \text{warm-up stage} \\ \frac{n_t - n_1}{n_2 - n_1} & \text{activation stage} \\ 1 & \text{full reward stage} \\ \frac{n_4 - n_t}{n_4 - n_3} & \text{annealing stage} \\ 0 & \text{final stage} \end{cases} \tag{11}$$

The construction of the graph and the training of the state connectivity model follow a bootstrapping process: in early episodes, the graph covers only a limited region of the environment, its edges are poorly calibrated, and the model is correspondingly biased. Therefore, during the warm-up stage, the graph is used to update the model, but the model is not used for policy learning (i.e., $\lambda = 0$), reflecting the limited reliability of the graph and the model.

In the activation stage, we linearly increase λ from 0 to 1, gradually introducing the dense signal induced by our model. This additional signal guides more efficient exploration and policy learning. λ is fixed to 1 afterwards, in the full reward stage.

In the annealing stage, since the dense reward is not potential-based (Ng et al., 1999), it not only alters the optimal policy but also changes the underlying optimization problem. To realign with the original task, we gradually phase out the dense signal by decreasing λ from 1 to 0. In the final stage, λ is fixed to 0.

3.5 Balancing between performance and cost

Several aspects of our method involve a tradeoff between performance and computational cost:

- **State sampling frequency:** Lowering the sampling frequency of candidate states for node features reduces comparisons during graph updates but may weaken the graph's representativeness.
- **Training data selection:** Training $\mathcal{C}_\theta$ on a subset of node pairs and edges speeds up training, at the potential cost of accuracy.

In the Experiments section, we report both the training cost and performance of G2QDR under these tradeoffs to identify the optimal balance point.

3.6 Algorithm: GCHRL + G2QDR

A detailed description of how the proposed G2QDR strategy can be integrated into online GCHRL algorithms is provided in Appendix D.

4 Experiments

As is standard in the GCHRL community, we evaluate our method across a range of long-horizon continuous control tasks simulated in the MuJoCo Todorov et al. (2012) environment. In this section, we employ gated fusion to construct an order-sensitive representation of state pairs.

4.1 Environment settings

We adopt the **AntMaze**, **AntGather**, **AntPush**, **AntFall**, and **Pusher** environments from MuJoCo, all configured with sparse reward settings. The first four involve complex navigation and manipulation tasks performed by a simulated multi-limbed robot, whereas **Pusher** focuses on object manipulation using a multi-jointed robotic arm. These environments exhibit varying degrees of asymmetry in their state transitions. **AntMaze** and **AntGather** are largely symmetric, as the agent can relatively easily return to previously visited states. In contrast, **AntPush**, **AntFall**, and **Pusher** exhibit asymmetry: reversing object displacement in AntPush and Pusher is considerably more difficult, and ascending cliffs in AntFall is substantially harder than descending them.

4.2 Comparative Analysis

We incorporated G2QDR in the following existing GCHRL methods:

- **HIRO** (Nachum et al., 2018b): One of the earliest approaches to demonstrate effective integration of goal-conditioned information in online hierarchical reinforcement learning.
- **HRAC** (Zhang et al., 2022): Extends HIRO by introducing an adjacency network that generates subgoals which are more readily reachable from the current state, thereby improving overall performance.
- **HESS** (Li et al., 2022): This method adds a regularization term over consecutive subgoal representations during each update to promote stability across episodes.
- **HLPS** (Wang et al., 2024): Uses a Gaussian process over subgoal representations to enable smoother and more consistent updates.

In addition to comparing these four G2QDR-augmented methods with their original counterparts, we also evaluate them against the following undirected-graph baseline to highlight the advantages of directed graph learning in quasimetric environments:

- **G4RL** (Zhang et al., 2025): A plug-in state connectivity estimator based on undirected graphs. It employs a graph encoder–decoder to learn the subgoal space and estimate transition plausibility from learned subgoal representations.

We report average success rates on **AntMaze**, **AntPush**, **AntFall**, and **Pusher**, along with the average number of objects collected in **AntGather** (Table 1). All results are averaged over 10 independent runs.

From Table 1, we observe that incorporating G2QDR into the base GCHRL methods consistently improves performance across environments. In particular, G2QDR-augmented methods not only achieve higher final success rates or scores but also reduce the variance of the baselines generally, with the most obvious gains appearing in more complex tasks. In quasimetric environments, the advantages of G2QDR over both the original baselines and the undirected G4RL are even more substantial. This suggests that G2QDR is especially effective at capturing and leveraging directional information in asymmetric settings.

Table 1: **Main results on MuJoCo environments across different methods.** For each method, we report the performance of the original baseline alongside its undirected variant (+G4RL) and directed variant (+G2QDR). All reported data points correspond to the final episode of training. Within each method group, the best-performing variant is underlined.

	AntMaze U-shape	**AntMaze W-shape**	**AntGather**	**AntPush**	**AntFall**	**Pusher**
HIRO	0.72 ± 0.09	0.53 ± 0.14	1.47 ± 0.18	0.05 ± 0.01	0.23 ± 0.04	0.17 ± 0.03
HIRO+G4RL	0.82 ± 0.07	0.61 ± 0.16	1.78 ± 0.10	0.14 ± 0.02	0.28 ± 0.04	0.21 ± 0.02
HIRO+G2QDR	0.89 ± 0.11	0.66 ± 0.09	1.96 ± 0.13	0.19 ± 0.04	0.35 ± 0.03	0.30 ± 0.03
HRAC	0.74 ± 0.05	0.59 ± 0.27	2.03 ± 0.22	0.10 ± 0.01	0.21 ± 0.06	0.24 ± 0.04
HRAC+G4RL	0.90 ± 0.03	0.68 ± 0.19	2.41 ± 0.26	0.12 ± 0.02	0.34 ± 0.03	0.22 ± 0.02
HRAC+G2QDR	0.87 ± 0.03	0.72 ± 0.16	2.54 ± 0.17	0.22 ± 0.02	0.29 ± 0.04	0.33 ± 0.03
HESS	0.80 ± 0.04	0.74 ± 0.08	2.74 ± 0.15	0.77 ± 0.10	0.55 ± 0.12	0.53 ± 0.15
HESS+G4RL	0.92 ± 0.02	0.84 ± 0.04	3.09 ± 0.21	0.74 ± 0.09	0.59 ± 0.12	0.56 ± 0.10
HESS+G2QDR	0.95 ± 0.03	0.81 ± 0.07	3.02 ± 0.13	0.85 ± 0.07	0.73 ± 0.11	0.68 ± 0.08
HLPS	0.78 ± 0.03	0.76 ± 0.11	2.96 ± 0.29	0.73 ± 0.09	0.71 ± 0.06	0.56 ± 0.17
HLPS+G4RL	0.94 ± 0.01	0.88 ± 0.04	3.11 ± 0.16	0.79 ± 0.02	0.64 ± 0.07	0.62 ± 0.07
HLPS+G2QDR	0.95 ± 0.03	0.83 ± 0.03	3.36 ± 0.20	0.86 ± 0.04	0.79 ± 0.05	0.77 ± 0.05

4.3 Ablation study

4.3.1 Impact of Auxiliary Reward Components

To evaluate the effectiveness of different auxiliary reward components within G2QDR, we consider the following variants:

- **a: High-level constraints only**: Equation (8) is applied to the high-level rewards, while the low-level significance is set to $\alpha_l = 0$ in equation (9).
- **b: Low-level correction only**: Equation (9) is applied to the low-level rewards, while the high-level significance is set to $\alpha_h = 0$ in equation (8).
- **c: Combined constraints and corrections**: Equations (8) and (9) are applied simultaneously to the high-level and low-level reward structures, respectively.
- **d: Full G2QDR (with penalty)**: Both equations (8) and (9) are applied as in variant c, supplemented by the additional penalty terms defined in equation (10).
- **Baselines (HIRO/HRAC/HESS/HLPS)**: Standard implementations of the vanilla baseline methods for performance comparison.

As before, all data reported in this section are averaged over 10 independent runs.

Table 2 presents the results of the ablation study across all evaluated environments and algorithms. Overall, combining high-level constraints and low-level corrections (variant (c)) consistently yields strong performance, often outperforming the use of either component in isolation. While the full G2QDR variant with penalty terms (d) further improves performance in several tasks, it does not uniformly dominate variant c, indicating that the effect of the penalty is environment-dependent.

In environments where Euclidean distance is a poor proxy for the true transition dynamics, for example, in **AntMaze**, where walls separate states that are close in Euclidean space but far in terms of transition distance, incorporating the low-level correction term leads to clear and consistent gains, highlighting its

Table 2: Ablation study of G2QDR variants. For each method, we report the performance of the original baseline together with its variants augmented by different types of auxiliary rewards (a–d). All results correspond to the final training episode. Within each method group, the best-performing variant is underlined.

	AntMaze U-shape	**AntMaze W-shape**	**AntGather**	**AntPush**	**AntFall**	**Pusher**
HIRO	0.72 ± 0.09	0.53 ± 0.14	1.47 ± 0.18	0.05 ± 0.01	0.23 ± 0.04	0.17 ± 0.03
HIRO-a	0.89 ± 0.11	0.63 ± 0.12	1.82 ± 0.14	0.14 ± 0.02	0.27 ± 0.02	0.23 ± 0.04
HIRO-b	0.81 ± 0.05	0.58 ± 0.15	1.56 ± 0.15	0.04 ± 0.01	0.26 ± 0.04	0.14 ± 0.02
HIRO-c	0.87 ± 0.08	0.66 ± 0.09	1.90 ± 0.11	0.12 ± 0.01	0.35 ± 0.03	0.26 ± 0.03
HIRO-d	0.80 ± 0.04	0.59 ± 0.07	1.96 ± 0.13	0.19 ± 0.04	0.31 ± 0.03	0.30 ± 0.03
HRAC	0.74 ± 0.05	0.59 ± 0.27	2.03 ± 0.22	0.10 ± 0.01	0.21 ± 0.06	0.24 ± 0.04
HRAC-a	0.81 ± 0.04	0.72 ± 0.16	2.39 ± 0.13	0.22 ± 0.02	0.19 ± 0.05	0.22 ± 0.04
HRAC-b	0.87 ± 0.03	0.69 ± 0.18	2.27 ± 0.26	0.12 ± 0.02	0.24 ± 0.03	0.26 ± 0.06
HRAC-c	0.84 ± 0.03	0.70 ± 0.15	2.54 ± 0.17	0.20 ± 0.03	0.29 ± 0.04	0.28 ± 0.02
HRAC-d	0.80 ± 0.04	0.63 ± 0.20	2.34 ± 0.21	0.19 ± 0.02	0.28 ± 0.03	0.33 ± 0.03
HESS	0.80 ± 0.04	0.74 ± 0.08	2.74 ± 0.15	0.77 ± 0.10	0.55 ± 0.12	0.53 ± 0.15
HESS-a	0.87 ± 0.03	0.76 ± 0.05	2.88 ± 0.27	0.74 ± 0.12	0.68 ± 0.16	0.60 ± 0.11
HESS-b	0.92 ± 0.02	0.81 ± 0.07	2.85 ± 0.10	0.79 ± 0.09	0.66 ± 0.08	0.47 ± 0.22
HESS-c	0.95 ± 0.03	0.80 ± 0.06	3.02 ± 0.13	0.82 ± 0.05	0.71 ± 0.13	0.58 ± 0.20
HESS-d	0.93 ± 0.04	0.78 ± 0.05	2.93 ± 0.24	0.85 ± 0.07	0.73 ± 0.11	0.68 ± 0.08
HLPS	0.78 ± 0.03	0.76 ± 0.11	2.96 ± 0.29	0.73 ± 0.09	0.71 ± 0.06	0.56 ± 0.13
HLPS-a	0.91 ± 0.02	0.79 ± 0.07	3.36 ± 0.20	0.79 ± 0.08	0.77 ± 0.05	0.66 ± 0.09
HLPS-b	0.88 ± 0.03	0.81 ± 0.05	3.20 ± 0.24	0.70 ± 0.06	0.69 ± 0.08	0.58 ± 0.11
HLPS-c	0.95 ± 0.03	0.83 ± 0.03	3.31 ± 0.14	0.78 ± 0.04	0.77 ± 0.06	0.77 ± 0.05
HLPS-d	0.90 ± 0.02	0.81 ± 0.07	3.03 ± 0.17	0.86 ± 0.04	0.79 ± 0.04	0.75 ± 0.07

role in improving local policy accuracy. The high-level constraint term also contributes to performance improvements by guiding the policy toward more feasible subgoals, although its effect is generally more pronounced when combined with low-level corrections.

The penalty term exhibits mixed behavior across tasks. It tends to improve performance in environments with asymmetric or partially irreversible dynamics, such as Pusher and AntPush, where it discourages transitions to states that are easy to reach but difficult to recover from. However, in more symmetric environments, the same penalty can restrict exploration and slightly degrade performance. These results suggest that, while the full G2QDR formulation is beneficial in certain settings, the combination of constraints and corrections alone provides consistently strong performance overall.

4.3.2 Balancing between time and performance

To assess the trade-off between computational efficiency and performance, we evaluate the three acceleration strategies introduced in Section 3.5. Specifically, we first vary the sampling frequency of candidate states used as node features, considering intervals of 1, 5, and 10 steps in HLPS for **AntPush** and **Pusher**.

All curves reported in Section 4 represent averages over 10 independent runs, with standard deviations shown as shaded regions. For clarity, all results are uniformly smoothed for visualization.

As shown in Figure 1, increasing the sampling interval substantially reduces computation time by reducing the number of graph interactions, while causing only minor degradation in success rates across both tasks.

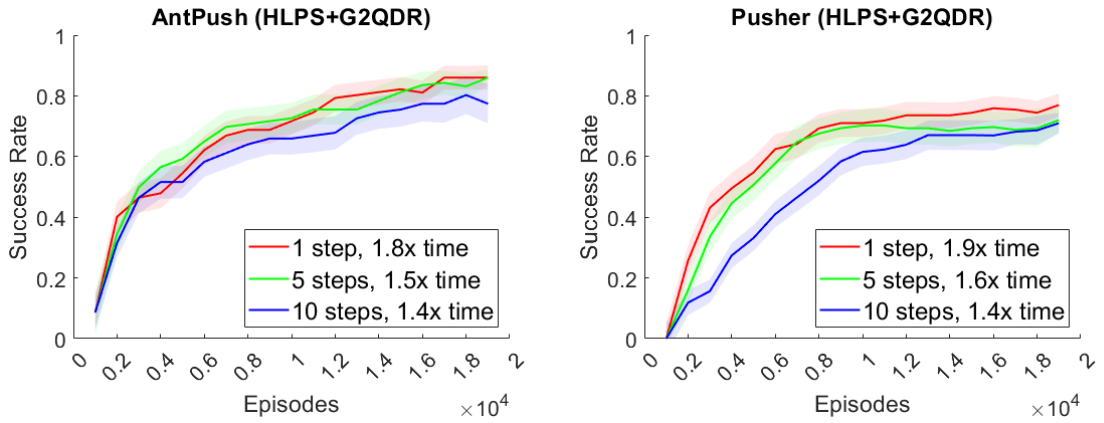

Figure 1: Success rates on (a) AntPush and (b) Pusher using HLPS+G2QDR. The number of steps in the legend denotes the chosen state sampling frequency, as described in Section 3.5, and the timescale is reported relative to the vanilla HLPS algorithm. Increasing the sampling interval substantially reduces computation time, with only minor degradation in success rates across both tasks.

Next, we vary the proportion of training data used for the state connectivity model, evaluating sizes of 50%, 75%, and 100% of all available training samples in the graph. The results in Figure 2 show that reducing the amount of training data yields only modest improvements in computational efficiency. In particular, decreasing the dataset size slightly shortens computation time, but the gains are limited. At the same time, the success rates on both AntPush and Pusher remain largely stable, with only minor degradation when using 75% of the available training samples. This suggests that G2QDR is relatively robust to moderate data reductions and does not rely heavily on the full dataset to achieve strong performance.

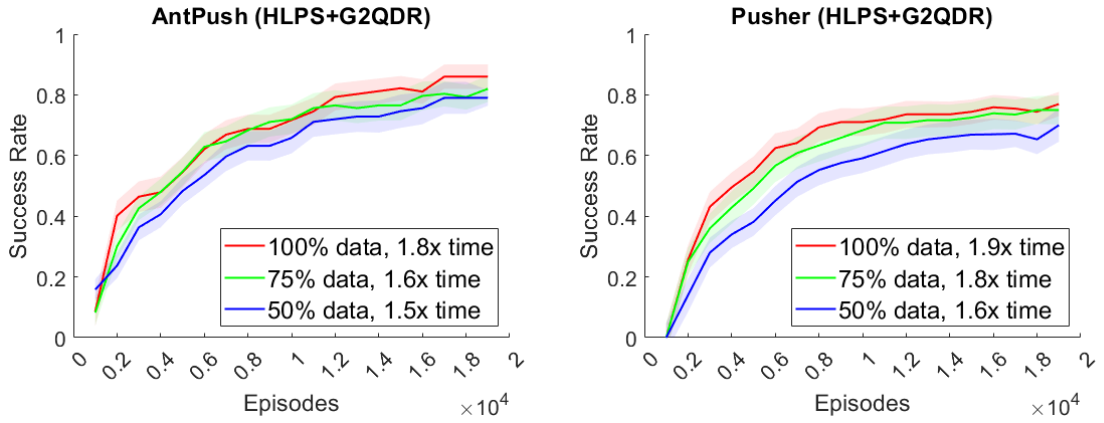

Figure 2: Success rate on (a) AntPush and (b) Pusher using HLPS+G2QDR. The percentages in the legend indicate the proportion of training samples used relative to the total number of edges in the graph, as described in Section 3.5. The timescale is computed with respect to the vanilla HLPS algorithm. Reducing the amount of data slightly decreases computation time; using 75% of the data results in only minor degradation in success rates across both AntPush and Pusher tasks.

These findings suggest that the primary computational bottleneck of G2QDR lies in the graph construction and node comparison procedures described in Section 3.1.1, rather than in the model training stage. Consequently, adjusting the sampling frequency emerges as an effective way to reduce computational cost, while largely preserving the performance gains provided by G2QDR integration.

4.4 Experimental analysis of dense signal schedule variations

This section examines how different dense reward schedules influence the performance of G2QDR. We introduce several variants of dense signal schedules, illustrated in Figure 3 (b), and present the corresponding success rate curves for **Pusher** using HLPS+G2QDR in Figure 3 (a).

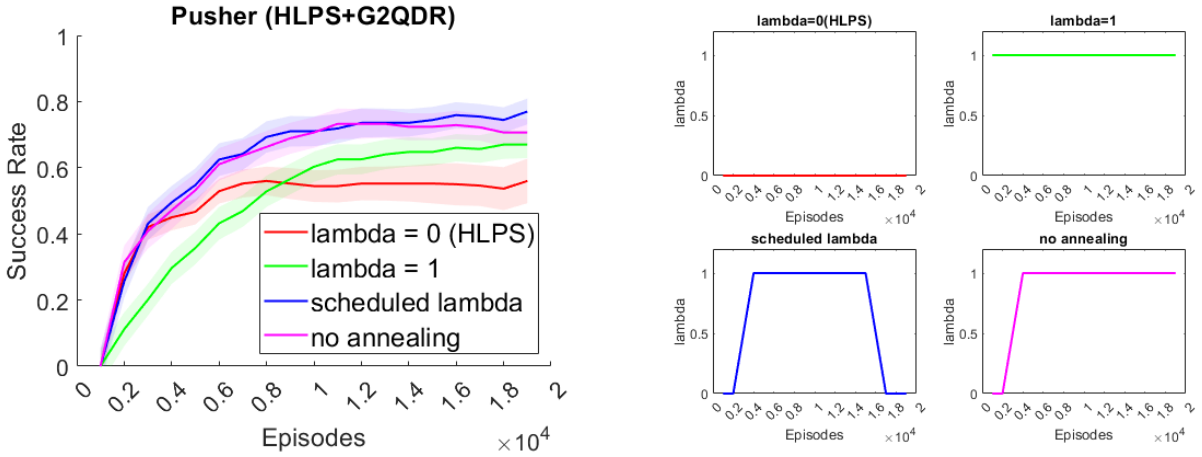

Figure 3: Ablation study of dense signal-strength scheduling. **Left:** Training curves under different λ schedules. **Right:** Evolution of the scheduling parameter λ across episodes.

From Figure 3, several key observations can be drawn. An immediate full reward setting ($\lambda = 1$) leads to slower convergence compared to other configurations, suggesting that instability in the state graph and the state connectivity model adversely affects both efficiency and convergence speed of the backbone method. Moreover, keeping λ high throughout later episodes results in inferior performance relative to variants that apply late-stage annealing. In contrast, a well-designed schedule for λ enables the architecture to achieve its full potential.

5 Conclusion

In this paper, we present a unified framework for graph-based GCHRL that incrementally constructs a directed state graph during exploration, learns a state connectivity model to estimate transition feasibility and evaluate newly encountered states, and leverages connectivity-based auxiliary rewards to enhance learning efficiency across multiple levels of agents.

Building upon the G4RL framework proposed by Zhang et al. (2025), our approach improves and extends its capabilities by explicitly modeling asymmetric state transitions, making it well-suited for quasimetric environments where transition dynamics are inherently directional and non-reversible. This enhancement allows the architecture to better capture realistic environment structures that are often overlooked by symmetric assumptions.

The proposed framework is broadly compatible with existing GCHRL methods, enabling seamless integration with a wide range of prior approaches. Empirical evaluations across diverse environments demonstrate consistent improvements over baseline methods, highlighting the effectiveness and robustness of our approach.

Overall, this work highlights the importance of modeling directional structure in state spaces for goal-conditioned hierarchical reinforcement learning. By translating state connectivity into auxiliary rewards, the proposed framework offers a flexible and scalable approach for tackling complex environments.

References

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.

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.

Zhang-Wei Hong, Tao Chen, Yen-Chen Lin, Joni Pajarinen, and Pulkit Agrawal. Topological experience replay. *arXiv preprint arXiv:2203.15845*, 2022.

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

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.

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.

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.

Bo Liu, Yihao Feng, Qiang Liu, and Peter Stone. Metric residual network for sample efficient goal-conditioned reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 8799–8806, 2023.

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.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287. Citeseer, 1999.

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.

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.

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.

Tongzhou Wang and Phillip Isola. On the learning and learnability of quasimetrics. *arXiv preprint arXiv:2206.15478*, 2022.

Tongzhou Wang, Antonio Torralba, Phillip Isola, and Amy Zhang. Optimal goal-reaching reinforcement learning via quasimetric learning. In *International Conference on Machine Learning*, pp. 36411–36430. PMLR, 2023.

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.

Vivienne Huiling Wang, Tinghuai Wang, and Joni Pajarinen. Hierarchical reinforcement learning with uncertainty-guided diffusional subgoals. *arXiv preprint arXiv:2505.21750*, 2025.

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.

Shuyuan Zhang, Zihan Wang, Xiao-Wen Chang, and Doina Precup. Incorporating spatial information into goal-conditioned hierarchical reinforcement learning via graph representations. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL `https://openreview.net/forum?id=a7Bx4s5gA8`.

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.

Deyao Zhu, Li Erran Li, and Mohamed Elhoseiny. Value memory graph: A graph-structured world model for offline reinforcement learning. *arXiv preprint arXiv:2206.04384*, 2022.

A Limitations and Future work

Despite the promising advantages demonstrated by G2QDR in hierarchical reinforcement learning (HRL) tasks, its performance remains highly sensitive to several key hyperparameters—such as ϵ_d, the learning rates (α), and the parameters p and w. Achieving strong and stable results across diverse environments, therefore, requires careful, and often time-consuming, manual tuning. In future work, we aim to mitigate this dependency by developing adaptive mechanisms that can automatically select or adjust these hyperparameters in response to environmental dynamics. Such approaches could improve robustness and generalization while significantly reducing the need for manual intervention. In addition, we plan to extend the scalability of G2QDR by optimizing its computational efficiency and evaluating its performance in larger, more complex environments, including those with higher-dimensional state and action spaces.

B Implementation details

B.1 Environment details

AntMaze-U Shape 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 1 if and only if the agent reaches within an Euclidean distance of 1 from the goal. 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 1 from the goal.

AntMaze-W shape This environment is a variant of Antmaze-U Shape. The size of the environment is 32×32. 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 $(2, 16)$.

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. The block can prevent the agent from reaching the goal if incorrectly pushed.

AntFall The size of the environment is 20×20. The agent need to reach the other end of a chasm by pushing a block into it as a bridge. If the agent falls into the chasm, it cannot recover and the episode effectively terminates.

Pusher The agent controls a 7-DoF multi-jointed robotic arm by applying continuous torques at each joint. The state space includes joint positions and velocities, as well as the positions of both the object and the target. The objective is to push the object to a designated target location.

B.2 Network architecture details

Our network architecture for the HRL agents follows prior work Nachum et al. (2018b); Zhang et al. (2022); Li et al. (2022); Wang et al. (2024). For HIRO, HRAC, HESS, and HLPS, both the high-level and low-level policies are implemented using TD3. The actor and critic networks in TD3 each consist of hidden layers with size 300.

For the state connectivity model, we employ a four-layer fully connected network with a hidden dimension of 128 and ReLU activation.

We use Adam as the optimizer for the actor and critic networks, as well as for the state connectivity model (Kingma & Ba, 2014).

C 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 3: 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
W	5
p	2
α_h	0.005
α_l	0.005
α_{hp}	0.01
α_{lp}	0.01

Table 4: Hyperparameters used in graph configurations and the state connectivity model.

D Algorithm

Now we describe our proposed method in Algorithm 1.

Algorithm 1 GCHRL+G2QDR

Require: High-level policy $\pi_h(g|\phi(s))$, low-level policy $\pi_l(a|\phi(s), g)$, replay buffer $\mathcal{B}$, state connectivity model $\mathcal{C}_\theta$, high-level action frequency K, window size W, significance hyperparameter α_h, α_l, α_{hp} and α_{lp}, adaptive signal strength λ, number of episodes N, nunber of steps in one episode T.

$n = 0$
while $n \leq N$ **do**
  $t = 0$
  while $t \leq T$ **do**
    if $t \mod K = 0$ **then**
      Execute the high-level policy $\pi_h(g_t|\phi(s_t))$ to sample the subgoal g_t.
    else
      Keep the subgoal g_t unchanged.
    Execute the low-level policy $\pi_l(a_t|\phi(s_t), g_t)$ to sample the atomic action a_t.
    Sample reward r_t and next state s_{t+1}.
    if $\lambda \neq 0$ **then**
      Calculate $r_h(s_t, g_t, s_{t+1})$, $r_l(s_t, g_t, a_t, s_{t+1})$, r_{hp} and r_{lp} using equations (8) , (9) and 10.
      Collect experience $(s_t, g_t, a_t, \lambda, [r_h, r_l, r_{hp}, r_{hp}], s_{t+1})$ and update the replay buffer $\mathcal{B}$.
    else
      Collect experience $(s_t, g_t, a_t, 1, [r_{\text{ext}}, -\|\phi(s_{t+1}) - g_t\|^2, 0, 0], s_{t+1})$ and update $\mathcal{B}$.
    Update node representations and edge weights using experiences and equations (1) to (4).
    Update state connectivity model $\mathcal{C}_\theta$ with node and edge information in the graph.
    $t = t + 1$
  Update low-level policy $\pi_l(a|\phi(s), g)$ using the chosen GCHRL algorithm with samples from $\mathcal{B}$.
  Update high-level policy $\pi_h(g|\phi(s))$ using the chosen GCHRL algorithm with samples from $\mathcal{B}$.
  Update λ according to n.
  $n = n + 1$