Hindsight Experience Replay with Evolutionary Decision Trees for Curriculum Goal Generation

Erdi Sayar[1](✉), Vladislav Vintaykin[1], Giovanni Iacca[2], and Alois Knoll[1]

[1] Technical University of Munich, Munich, Germany
{erdi.sayar,vladislav.vintaykin}@tum.de, knoll@mytum.de
[2] University of Trento, Trento, Italy
giovanni.iacca@unitn.it

Abstract. Reinforcement learning (RL) algorithms often require a significant number of experiences to learn a policy capable of achieving desired goals in multi-goal robot manipulation tasks with sparse rewards. Hindsight Experience Replay (HER) is an existing method that improves learning efficiency by using failed trajectories and replacing the original goals with hindsight goals that are uniformly sampled from the visited states. However, HER has a limitation: the hindsight goals are mostly near the initial state, which hinders solving tasks efficiently if the desired goals are far from the initial state. To overcome this limitation, we introduce a curriculum learning method called HERDT (HER with Decision Trees). HERDT uses binary DTs to generate curriculum goals that guide a robotic agent progressively from an initial state toward a desired goal. During the warm-up stage, DTs are optimized using the Grammatical Evolution algorithm. In the training stage, curriculum goals are then sampled by DTs to help the agent navigate the environment. Since binary DTs generate discrete values, we fine-tune these curriculum points by incorporating a feedback value (i.e., the Q-value). This fine-tuning enables us to adjust the difficulty level of the generated curriculum points, ensuring that they are neither overly simplistic nor excessively challenging. In other words, these points are precisely tailored to match the robot's ongoing learning policy. We evaluate our proposed approach on different sparse reward robotic manipulation tasks and compare it with the state-of-the-art HER approach. Our results demonstrate that our method consistently outperforms or matches the existing approach in all the tested tasks.

Keywords: Decision Tree · Reinforcement Learning · Curriculum Learning · Sparse Reward · Multi-goal Tasks

1 Introduction

Reinforcement learning (RL) is a well-known computational paradigm for discovering optimal actions through trial and error, so as to maximize rewards without explicit guidance [1]. In recent years, deep RL, which combines RL with deep neural network (DNN)-based function approximators, has made remarkable advancements and achieved impressive outcomes, exceeding human-level performance e.g. in playing Atari games [2,3], beating Go champions [4], and solving robotic tasks [5–8]. In these scenarios, the design of an effective reward function [9] is one of the most challenging aspects. This is because the reward function must be carefully tailored to the specific task at hand, and it must be able to capture the desired behavior of the agent. However, in many cases, the admissible behavior of the agent is not known in advance, which makes it difficult to design an effective reward function. As a result, binary rewards are often used in RL algorithms. This is particularly true for robotic tasks, where binary rewards are used to simply indicate task success or failure, offering a comparatively simpler alternative to the intricate design of more complex reward functions. However, as binary rewards only provide information about whether the task is completed or not, without providing more detailed information about the actual agent's progress towards the goal, RL algorithms encounter difficulties at learning effectively [10]. To address the challenge of sparse rewards, Hindsight Experience Replay (HER) [11] offers a promising solution. HER substitutes the *desired goals* with the *achieved goals*, which are sampled uniformly from the visited states. In this way, HER can convert failed episodes into successful ones. However, a limitation of HER lies in its disregard for the importance of the states visited during the learning process, resulting in sample inefficiency. Different methods have been introduced to address the issue of sample inefficiency in RL, e.g. through prioritization of the replay buffer, such as Energy-Based Prioritization [12], which prioritizes experiences with higher energy, and Maximum Entropy-based Prioritization [13], which samples replay trajectories more frequently based on entropy.

Another challenge in deep RL is its explainability. In this regard, the growing interest in explainable RL [14] has been driven by the general lack of interpretability of deep RL, being built on top of opaque DNN [15]. Interpretability can be achieved by using interpretable models, such as Decision Trees (DTs), which can be highly interpretable, as long as they are shallow [16]. To interpret an RL agent with a DT, various approaches have been proposed. Coppens et al. [17] proposed distilling the output of a pre-trained deep RL policy network into a "soft" DT. Another method, called VIPER [18], uses a policy extraction technique to convert a complex, high-performing DNN-based policy into a simpler DT-based policy. Ding et al. [19] used Cascading DTs (CDTs), where the feature learning tree is cascaded with a decision-making tree. Roth et al. [20] proposed Conservative Q-Improvement (CQI) to produce a policy in the form of a DT, resulting in smaller trees compared to existing methods, without sacrificing policy performance. Hallawa et al. [21] proposed a methodology based on Genetic Programming (GP) to produce Behavior Trees (BTs) combined with various forms of RL such as Q-learning, DQN, and PPO. Other authors extended this approach by using Grammatical Evolution (GE) to produce DTs in combination with Q-learning performing online learning on the leaves of the tree, and testing the resulting agents on various RL tasks, including tasks with discrete action spaces [22–24], tasks with continuous action spaces [25], and multi-agent tasks [26]. However, most of these approaches focus on the use of DTs for defining the policy of the agent. To the best of our knowledge, no previous work has used DTs to generate curriculum goals during the training process, hence guiding the agent to the desired goal even in contexts with sparse rewards. This is precisely the main focus of the present work. Our hypothesis is that by using DTs, it is possible to generate curriculum waypoints in a straightforward and interpretable way, hence facilitating the agent's navigation and task solving, as opposed to directly generating control commands to the agent using DTs. Here, we propose a curriculum learning approach based on DTs, that we dub Hindsight Experience Replay with Decision Trees (HERDT). Our methodology works as follows. Initially, we transform a robotic environment into a straightforward grid representation, comprising an initial position and a desired goal. Subsequently, we employ GE to optimize the DT structures, which are then further optimized by means of Q-learning, as done in [22]. By doing so, we ensure that the DTs effectively capture the environment's characteristics and decision-making process. Then, we leverage the optimized DTs to generate curriculum goals that serve as navigational waypoints for guiding the agent through the environment. In the experimentation, we compare our method with the baseline HER on standard multi-goal RL benchmark tasks and perform thorough ablation studies. To summarize, the main contributions of this paper are the following:

- We propose HERDT, a curriculum learning approach composed of two stages, namely: (1) A *warm-up stage*, where we construct a grid environment with the same initial and desired positions as in the robotic environment. We then use the GE algorithm to optimize binary DTs with the grid environment. As a result, DTs acquire capabilities to provide guidance to an agent in the robotic environment. (2) A *training stage*, where we sample curriculum goals from those optimized binary DTs to guide the robotic agent toward the desired goal. Since binary DTs output discrete values given an input state, we further fine-tune these curriculum points to adjust their difficulty and ensure that they are neither too easy nor too hard for the robot's ongoing learning policy.
- We compare our approach against HER on a set of benchmark tasks with the 7-DOF Fetch Robotic-arm MuJoCo simulation environment [27].
- In the ablation studies, we investigate the impact of recursively generated curriculum goals and the effect of the fine-tuning of the curriculum points on the success rate of the tasks at hand.

The rest of the paper is structured as follows. The next section provides the background concepts. Then, Sect. 3 describes our proposed method. Section 4 presents the numerical results. Finally, Sect. 5 provides the conclusions.

2 Background

Hindsight Experience Replay (HER). As mentioned earlier, RL methods often struggle to explore the environment effectively, especially under sparse reward conditions. HER [11] addresses this sample efficiency problem by enabling agents to learn from their failures. When an agent fails to achieve its desired goal, HER replays the experience as if the agent had achieved a different and achievable goal. By doing so, the agent can receive a reward and learn at least how to accomplish a task from the achieved states. In this way, HER converts failed episodes into successful ones.

GE with Q-Learning. According to the GE with Q-learning approach proposed in [22], a population of genotypes, each one encoded as a fixed-length list of codons, is evolved. During the evaluation, each genotype is converted into a phenotype, which represents the policy of the agent based on a binary DT. Then, the agent acts accordingly in the environment and receives a reward signal. A Q-value is calculated from the reward signal, using the Q-learning approach [28], and the cumulative reward is used as a fitness value for selecting the individuals in GE (here, an "individual" refers to a DT). Then, a standard one-point crossover operator is applied, which simply sets a random cutting point and creates two individuals by mixing the two sub-strings of the genotype. This means that individuals whose genes are not expressed in the phenotype are not pruned. Then, a classical uniform mutation operator mutates each gene according to a given probability. In this way, DT policies can be optimized to perform optimal actions leading to receiving a high reward from the environment.

Multi-goal RL. Multi-goal RL, in which an agent learns to achieve multiple goals sampled from a goal distribution, can be modeled as a goal-directed Markov decision process with continuous state and action spaces $\langle \mathcal{S}, \mathcal{A}, \mathcal{G}, \mathcal{T}, r, p, \gamma \rangle$, where $\mathcal{S}$ is a continuous state space, $\mathcal{A}$ is a continuous action space, $\mathcal{G}$ is a goal distribution (indicating with g a desired goal sampled from $\mathcal{G}$), $\mathcal{T}(s'|s,a)$ is the transition function, $r(s,g)$ denotes the immediate reward obtained by an agent upon reaching state $s \in \mathcal{S}$ given goal g, $p(s_0, g)$ is a joint probability distribution over initial states and desired goals, and $\gamma \in [0,1]$ is a discount factor. Specifically, the reward function is defined as:

$$r(s,g) = \mathbf{1}[\|\phi(s) - g\|_2 \leq \epsilon] - 1, \tag{1}$$

where $\mathbf{1}$ is the indicator function, ϕ is a predefined function that maps a state to the achieved goal[1], and ϵ is a fixed threshold. In this context, the learning task can be modeled as an RL problem that seeks a policy $\pi : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$, with the primary objective of maximizing the expected discounted sum of rewards for any given goal.

While HER can be applied with various off-policy RL algorithms, in this work we opt for utilizing DDPG [29], in alignment with the HER setting presented in

[1] The physical interpretation of the achieved goal depends on the task at hand. For some robotic manipulation tasks, the robot needs to pick and place (Fig. 5b), push (Fig. 5c), or slide (Fig. 5d) an object. In this case, the achieved goal corresponds to the x-y-z position of the object. Conversely, if there is no object in the task (Fig. 5a), the achieved goal is defined as the position of the end-effector of the robot.

[30]. DDPG is an off-policy actor-critic algorithm that consists in a deterministic policy $\pi_\theta(s,g) : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$, parameterized by θ, and a state-action value function $Q_\phi(s,a,g) : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$, parameterized by ϕ. Gaussian noise with zero mean ($\mu = 0$) and constant std. dev. is added to the deterministic policy π_θ to improve exploration. The behavior policy, π, is then used for collecting the results over the episodes:

$$\pi(s,g) = \pi_\theta(s,g) + \mathcal{N}(\mu, \sigma^2). \tag{2}$$

The Q-value is trained by minimizing the Temporal Difference (TD) error, which is defined as the following loss function:

$$\mathcal{L}_{critic} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1},g)\sim\mathcal{B}}\left[(y_t - Q_\phi(s_t,a_t,g))^2\right] \tag{3}$$

where t is the timestep and $y_t = r_{t+1} + \gamma Q_\phi(s_{t+1}, \pi_\theta(s_{t+1}, g), g)$. Subsequently, the policy π is updated using policy gradient on the following loss function:

$$\mathcal{L}_{actor} = -\mathbb{E}_{(s_t,g)\sim B}\left[(Q(s_t, \pi_\theta(s_t,g), g))\right]. \tag{4}$$

As Q-function, we use the universal value function (UVF), which is defined as $Q_\phi(s_t, a_t, g)$, where s_t is the current state, a_t is the current action, g_t is the goal, and π is the policy. The UVF is a generalization of the traditional value function that integrates the goal into the value estimation and it has been shown to be able to successfully generalize to unseen goals, which makes it a promising approach for multi-goal RL [31]. The UVF estimates the expected return from state s under policy π, given goal g.

3 Proposed Method

Our proposed HERDT method builds on some of the aforementioned related works and consists of two main parts:

- *Warm-up stage*: before starting to train the robotic agent, as in [22] we optimize binary DTs with GE while using Q-learning to learn the state-action value of the grid environment constructed starting from the robot environment, see Fig. 1. The first environment is a discrete representation of the latter, including the initial position and target position.
- *Training stage*: during this stage, the current state of the robot is provided to the DTs in each episode, and the next curriculum goals are sampled accordingly. Since the DTs produce discrete values, we refine these generated curriculum points by adjusting them based on the Q-value of the robotic agent. This refinement ensures that these points are appropriately calibrated, i.e., that they are neither overly simplistic nor excessively challenging for the robotic agent during the learning process.

The conversion to the grid environment is motivated by two primary factors. Firstly, evolving a DT in a continuous environment using GE would be too computationally intensive. Secondly, employing a binary DT to generate discrete curriculum goals is particularly suitable for grid environments.

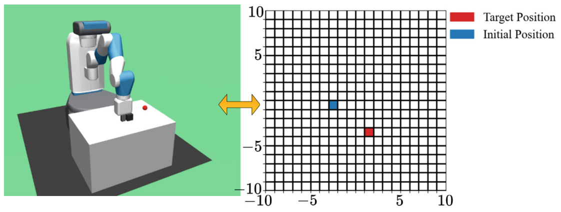


Fig. 1. Constructing a grid environment (right) from a robotic environment (left).

Table 1. Grammar rules used to evolve DTs. The symbol "|" indicates the option to select from various symbols. "comp_op" is a concise term for "comparison operator", with "lt" and "gt" representing "less than" and "greater than" operators, respectively. "input_var" indicates one of the potential inputs. It is important to emphasize that each input variable is associated with the same set of constants.

Rule	Production
dt	$< if >$
if	$if < condition > then < action > else < action >$
condition	$input\_var < comp\_op >< const >$
action	$leaf \| < if >$
comp_op	$lt \| gt$
const	$[0, 20)$ with step 1

3.1 Optimizing DTs

The grid environment is constructed using the same initial and target positions as the robotic task. As mentioned earlier, some robotic tasks, such as pushing and sliding, inherently require that robots move objects only on a flat table surface. These can be effectively modeled using a 2D grid environment, as curriculum points are generated only on the table. In contrast, tasks like reaching and pick-and-place, require the robot to navigate in a 3D space and as such they can be modeled using a 3D grid environment. We assume that we have an agent in the grid environment that can only move forward, backward, right, and left (in addition, up and down in the 3D case). For the grid environment, we define the following reward function:

$$r(s,g)_{grid} = \begin{cases} 1 & \text{if goal is reached} \\ \frac{-1}{10\, s-g||_2} & \text{otherwise} \end{cases} \quad (5)$$

where s and g are the agent state and target position, respectively. The DTs are evolved with the grammar rules given in Table 1, as described in [22,32]. The genotype of an individual is encoded as a fixed-length list of codons, which are represented as integers. The genotype is then translated to the corresponding phenotype, which is a DT. The policy $a_t = \pi_{DT}(\cdot)$ encoded by the tree is executed and the reward signals are obtained from the grid environment for each timestep t. Then, the average reward $R_t = \frac{1}{n}\sum_t^n r_t$ is used as fitness value for GE and at the same time the Q-values of the DT's leaves are updated using the Q-learning approach [28]:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right). \tag{6}$$

Then, the action for each leaf is selected as $\arg\max_a Q(s, a)$. Each action corresponds to a fixed-step movement on the grid environment. These actions are employed as curriculum goals during the robot's training and are scaled by the Q-value of the critic network to ensure they are appropriately challenging, without being overly difficult for the robot. In the selection process, a parent is replaced if its offspring has a better fitness value. The standard one-point crossover operator is used as a crossover mechanism, randomly selecting a cutting point and generating two individuals by mixing the two sub-strings of the genotype. A standard uniform mutation operator is utilized to randomly select a new value from within the variable's range of variation. This process is visually illustrated in Fig. 2.

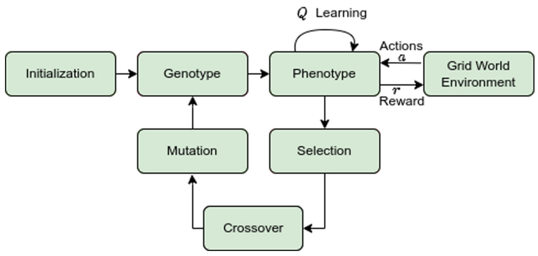


Fig. 2. Evolutionary process for the DTs.

Table 2. DT actions.

$a = \pi_{DT}$	Description
0	+x direction
1	−x direction
2	+y direction
3	−y direction
4	+z direction
5	−z direction

3.2 Generating Curriculum Goals

As seen, the actions generated by the DT policy in the grid environment serve as intermediate goals for guiding the robot toward the final goal. These actions are discrete values that describe the next step to move into the grid environment, thus determining the curriculum points for the robot.

After optimizing the DT policies, we provide the current state of the robot, s_t, to the state-to-goal mapping function $\phi(s)$, which is given to the DT policy, which in turn outputs a discrete value between 0 and 5 (0 and 3 in 2D) as the action. All the actions and their corresponding definitions are listed in Table 2. Each value corresponds to the next step in the grid environment. For example, the action 0 means that the robot should move an object to the next position in the positive x dimension in the grid environment.

The action (i.e., the next target) of the DT policy depends on the grid size. Specifically, if the grid size is small, then the next target will be closer to a task location that has already been reached by the robot. However, this can lead to a slower optimization process for the DTs. Consequently, the robot cannot efficiently improve its policy further as it has already explored the area around its current position. On the other hand, if the grid size is large, then the next target will be far from the robot's current position. This can lead to faster optimization of the DT, but it may not be optimal for the robot as it will not guide the robot efficiently.

Because of these reasons, we multiply the action output from the DT policy by a feedback value (i.e., the Q-value from the critic network) so that we can adjust the next target position to be neither too difficult nor too easy for the robot. An example of curriculum goals generated throughout the entire training process is shown in Fig. 3, with colors ranging from red to blue representing the curriculum goals across different episodes of the training.

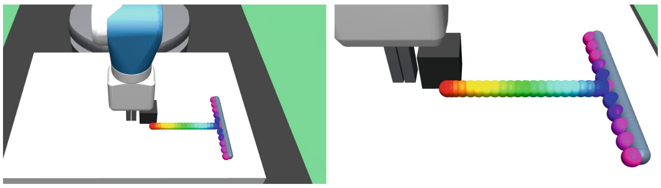


Fig. 3. Visualization of the generated curriculum goals in the FetchPush task. The colors (ranging from red to light blue) represent the curriculum goals across different episodes of the training. The gray color represents the desired positions. (Color figure online)

The overall HERDT algorithm is given, in the form of pseudo-code, in Algorithm 1.

In the warm-up stage (lines 7–14), as explained in Sect. 3.1, the DT policy generates discrete actions, such as 'up' or 'down', based on the corresponding grid location of the given achieved goal. These actions are interpreted according to Table 2. By integrating the action from the DT policy into the current achieved goal, we can determine the subsequent goal for the robotic agent to pursue. Since the DT policy outputs discrete actions, directly adding these actions to the current achieved goal might lead to abrupt jumps from one goal to the next one, which may be hard for the robot to achieve. To address this, we introduce a refinement step in line 18, where κ (the curriculum goal) is multiplied by a factor proportionate to the Q-value. The Q-value represents the expected return for the robot when it starts from a specific state and acts according to the policy. A higher Q-value indicates a higher expected reward for that state, suggesting that the robot has effectively learned that particular state-action pair. As a result, in situations where the Q-value is high, we infer that the robot has learned the state well, and we adjust the next desired goal to be slightly farther away compared to scenarios with lower Q-values. We can achieve this by calculating $\nu = \kappa \cdot |c - Q'|$, where ν indicates the adjusted curriculum goal, c is a hyperparameter, and Q' is defined as follows:

$$Q' = \begin{cases} -1 & \text{if } Q < -1 \\ 0 & \text{if } Q > 0 \\ Q & \text{otherwise.} \end{cases} \tag{7}$$

For each timestep, the current policy π of the robotic agent (the actor neural network) outputs the action a_t, given the current state s_t and the desired goal ν (line 20), which is then executed in the robotic environment, and the next state and reward are received (line 21). Subsequently, the experiences $(s_t, a_t, r_t, s_{t+1}, \nu)$ are collected and added to the replay buffer (line 22). If the achieved state is regarded as a hindsight goal (HER idea), then the reward is recalculated and added to the replay buffer (lines 23–25). A minibatch is sampled from the replay buffer. The Q-value function and the policy π are updated using minibatch in the loss functions Eq. (3) and Eq. (4), respectively (lines 29–30). In the test rollout, we calculate the success rate by setting the target goal and

Algorithm 1. Hindsight Experience Replay with Decision Trees (HERDT)

```
1: Input: no. of iterations $K$, no. of epochs $L$, no. of episodes $M$, no. of timesteps $T$, no. of test
   rollouts $n_{test}$, $\epsilon$
2: Select an off-policy algorithm $\mathbb{A}$                          ▷ In our case $\mathbb{A}$ is DDPG
3: Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$
4: Sample initial state $s_0$ and target goal $g$ from robotic environment
5: Construct grid environment
6: Initialize DTs with the grammar rules in Table 1 and randomly assign values to leaf nodes
7: for t = $1 \ldots K$ do                                               ▷ Warm-up stage
8:     Encode genotype of a DT
9:     Translate the genotype into a phenotype $a_{DT} = \pi_{DT}(s)$
10:    Execute the action $a_{DT}$ in the grid environment
11:    Obtain the next state $s_{t+1}$ and reward $r_t$
12:    Update $Q$-value in Eq. (6) and use it as fitness value for GE
13:    Select individuals, apply crossover and mutation operators
14: end for
15: for epoch = $1 \ldots L$ do                                          ▷ Training stage
16:     for episode = $1 \ldots M$ do
17:         Sample state $s$ and curriculum goal $\kappa = \phi(s) + \pi_{DT}(\phi(s))$
18:         $\nu = \kappa \cdot |c - Q'|$                                 ▷ Eq. (7)
19:         for $t = 1 \ldots T$ do                                       ▷ Rollout episode
20:             $a_t = \pi(s_t, \nu)$
21:             Execute the action $a_t$, obtain the next state $s_{t+1}$ and reward $r_t$
22:             Store transition $(s_t, a_t, r_t, s_{t+1}, \nu)$ in replay buffer $\mathcal{B}$
23:             Sample additional goals from achieved states for replay $G := \mathcal{S}(\text{episode})$
24:             for $g' \in G$ do                                         ▷ Hindsight goal
25:                 Recompute reward $r'_t$
26:                 Store transition $(s_t, a_t, r'_t, s_{t+1}, g')$ in replay buffer $\mathcal{B}$
27:             end for
28:         end for
29:         Sample a minibatch $b$ from replay buffer $\mathcal{B}$
30:         Update $Q$ and $\pi$ with $b$ to minimize $\mathcal{L}_{critic}$ in Eq. (3) and $\mathcal{L}_{actor}$ in Eq. (4)
31:     end for
32:     $success\_rate \leftarrow 0$
33:     for $t = 1 \ldots n_{test}$ do                                   ▷ Test rollouts
34:         $a_t = \pi(s_t, g)$
35:         Execute the action $a_t$, obtain a next state $s_{t+1}$ and reward $r_t$
36:         if $\|\phi(s_{t+1}) - g\|_2^2 \leq \epsilon$ then                ▷ Eq. (1)
37:             $success\_rate \leftarrow success\_rate + 1/n_{test-rollouts}$
38:         end if
39:     end for
40: end for
```

evaluating how many rollouts out of n_{test} successfully achieve the target goal (lines 32–39).

4 Experimental Results

In this section, we assess the proposed HERDT approach by experimenting with a 7-degree-of-freedom robotic arm [33] within the MuJoCo simulation environment [27]. Robotic manipulation tasks are indeed well-established benchmark tasks for multi-goal RL [30]. In this study, four standard manipulation tasks (FetchReach, FetchPickAndPlace, FetchPush, and FetchSlide) are selected, as shown in Fig. 5 and the selected tasks are briefly described below:

– *FetchReach*: This task consists of controlling the robotic arm to move its gripper to a target position in a 3D space (shown as a red dot in Fig. 5a).
– *FetchPickAndPlace*: This task consists of controlling the robotic arm to grasp the black object with its gripper and place it at the target position in a 3D space (shown as a red dot in Fig. 5b).
– *FetchPush*: This task consists of controlling the robotic arm to push the black object with its clamped gripper to the target position in a 2D space (shown as a red dot in Fig. 5c).
– *FetchSlide*: This task consists of controlling the robotic arm to make the black object slide along the sliding table from its initial position to the target position in a 2D space (shown as a red dot in Fig. 5d).

In the warm-up stage, we optimize the DTs using the GE algorithm. These optimized DTs are employed to generate curriculum goals during the training stage for the robotic tasks. For example, we illustrate an optimized DT and its corresponding grid word environment for the FetchPush task in Fig. 4. The mechanism behind how DTs generate curriculum goals during training-stage is explained below: As an example, the initial and target positions are indicated as (x = 8, y = 10) and (x = 9, y = 16) in Fig. 4, respectively. For each episode, the initial position (x = 8, y = 10) is given to the DT, and node 0 checks whether x is lower than 9 or not. In our case, x = 8, so it is lower than 9, then the DT goes to node 1. This node checks if y is greater than 11 (in our case, it is not), and then it selects node 4, which checks y. Since y in our case is lower than 13, the DT selects node 7, which again checks y. In our case y is greater than 7, then node 9 is selected, which outputs action 0, i.e., movement in the +x direction. So, our next curriculum goal will be in (x = 9, y = 10). If this position is given to the DT, then node 0 will check again x. In this case, x is not lower than 9. Then, the DT selects node 2, which outputs action 2, corresponding to movement in the +y direction. Our next curriculum goal will be in (x = 9, y = 11) and after that position, the DT will always select action 0, which results in arriving at the target position after 6 iterations. The other action (2) is the movement in the −y direction as specified in Table 2.

After every epoch, the performance is evaluated by running 100 deterministic test rollouts. Then, the success rate of the tests is calculated by averaging the results, as detailed between lines 32 and 39. In the selected tasks, the state vector incorporates multiple components, including the target position and the current position, orientation, linear velocity, and angular velocity of both the end-effector and the object. The goal represents the desired position for an object. If an object reaches the target within a threshold distance, as defined in Eq. (1), then the task is considered successfully completed and the agent obtains a 0 reward value. If the object falls outside the defined range of the goal, the agent is penalized with a negative reward of −1. We test the proposed approach with 10 different seeds for all the experimental settings.

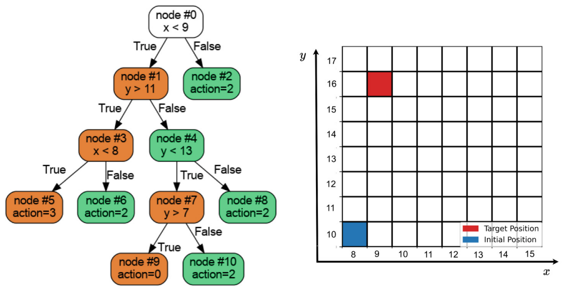
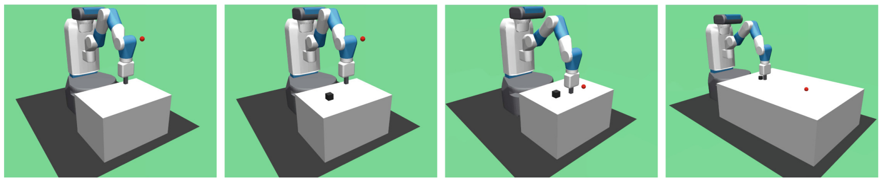


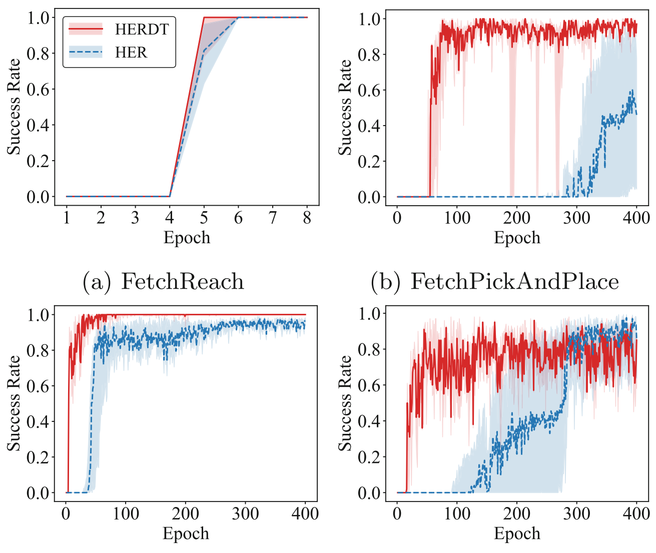
Fig. 4. An example DT generated for the FetchPush task (left) and its convergence graph (right).



(a) FetchReach (b) PickAndPlace (c) FetchPush (d) FetchSlide

Fig. 5. Overview of the robotic manipulation benchmark tasks.

The trend of the test success rate, shown for all environments in Fig. 6, clearly shows the impact of generating curriculum goals on the success rate. From the figures, it can be evidently inferred that HERDT outperforms HER (in terms of success rate and convergence) across all the considered robotic manipulation environments, except for the FetchSlide. It should be noted in fact that, for the FetchSlide task, HERDT converges faster and has a lower standard deviation than HER. However, the success rate of HER becomes slightly better than our method in the final episodes of the FetchSlide task. HERDT converges to the near-optimal policy after 5, 75, 35, and 50 epochs, while HER requires 6, 400, 65 and 300 epochs to converge for FetchReach, FetchPickAndPlace, FetchPush, and FetchSlide, respectively.



(a) FetchReach (b) FetchPickAndPlace

(c) FetchPush (d) FetchSlide

Fig. 6. Test success rate for the tasks. The mean success rate (line) and inter-quartile range (shaded area) are shown for 10 random seeds.

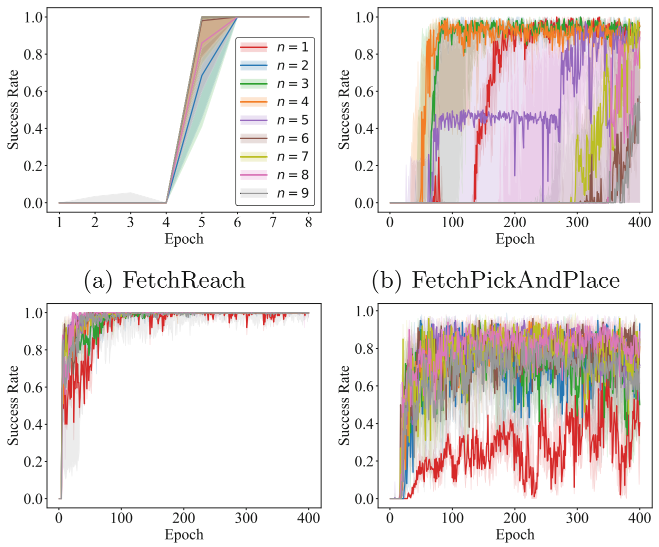
This can be attributed to the fact that our approach, HERDT, employs a DT that generates curriculum goals, guiding the agent step by step from its initial position to the target goal. In other words, generating curriculum goals decomposes the task into simpler sub-tasks. In contrast, HER directly instructs the robot to reach the target goal without providing any curriculum goals.

4.1 Ablation Studies

Experiments with Curriculum Goals. As discussed earlier, in our method the DT policy takes the achieved goals as input by converting the state s_t using the state-to-goal mapping function $\phi(s_t)$. In the first ablation study, we then aim to answer the following question: What if we recursively feed the generated goal g_t into the DT policy back again, such that $g_{t+1} = g_t + \pi_{DT}(g_t)$, and repeat this process n times such that $g_{t+n} = \frac{1}{n}\sum_{i=1}^{n} g_i$? How would this affect the performance of the agent, compared to generating the curriculum goal g_t and using it as the next desired goal during the next episode?

The idea behind this experiment is that we can generate the next curriculum goals g_{t+1} given the previous curriculum goal g_t, recursively. Such curriculum goals represent positions in a 3D space. We can repeat this process n times and obtain different future curriculum goals $g_{t+1}, g_{t+2}, \ldots, g_n$. However, these goals might be very far away from g_t, making the task excessively challenging for the robot. Additionally, some generated goals might lie outside the designated task environment. To mitigate this issue, we scale the recursively generated curriculum goals by multiplying their sum by $1/n$, where n is the number of curriculum goals.

Figure 7 illustrates the impact of the parameter n on the success rate. It can be inferred that when $n = 1$, the performance dwindles in the FetchPickAndPlace, FetchPush, and FetchSlide tasks. This is because curriculum goals approach the desired goals slowly and may not cover the entire distribution of sampled desired goals within the given number of training epochs. On the other hand, as n increases, the curriculum goals approach the desired goals more quickly. Yet, these curriculum goals might be too challenging for the robot to achieve, resulting in a lower success rate at the beginning of the training in all tasks except for the FetchPush task, where the parameter n does not significantly affect the outcomes.



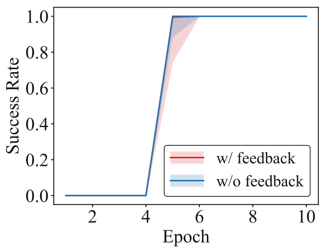
(a) FetchReach (b) FetchPickAndPlace

(c) FetchPush (d) FetchSlide

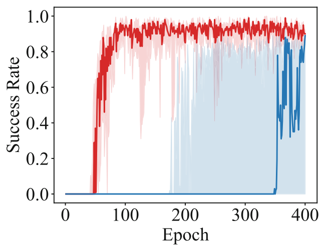
Fig. 7. Through this ablation study, we analyze the impact of recursively generated curriculum goals on the success rate of robotic manipulation tasks. The mean success rate (line) and inter-quartile range (shaded) are shown for 10 random seeds.

Experiments Without Feedback from *Q*-Value. In the second ablation study, we aim to answer the following question: How does the feedback from the Q-value (from the critic network) affect the success rate?

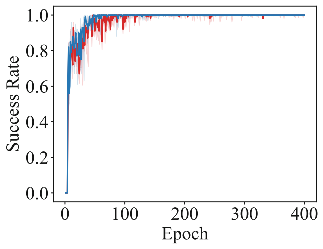
As discussed earlier, after optimizing the binary DTs, we modify the curriculum goals generated during the training process by multiplying them by the Q-value. This adjustment is made to ensure that their positions are optimized accordingly, making them neither too challenging nor too easy for the robot to



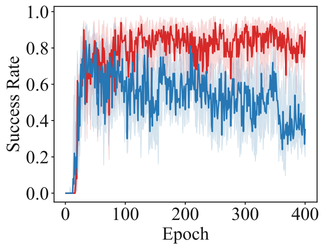
(a) FetchReach



(b) FetchPickAndPlace



(c) FetchPush



(d) FetchSlide

Fig. 8. Through this ablation study, we analyze the impact of the feedback on the success rate for the robot manipulation tasks. The average success rate (line) and inter-quartile range (shaded) are shown for 10 random seeds.

reach. Figure 8 depicts the impact of feedback on the success rate in benchmark tasks. Based on the simulation results, it becomes evident that the feedback value plays a crucial role in enhancing learning performance and fine-tuning the generated curriculum points to align with the robot's ongoing learning policy.

5 Conclusion

In this study, we introduced a novel curriculum learning RL method, HERDT, which leverages binary Decision Trees (DTs) to generate curriculum goals for the robotic agent. Prior to training the robotic agent, a *warm-up* phase is performed, during which DTs are optimized using the Grammatical Evolution (GE) algorithm. In the training stage, the optimized DTs generate curriculum goals to guide the agent toward the target goal. Since DTs output discrete values, we further refine these curriculum points based on feedback from the robotic agent. This fine-tuning helps the robotic agent find the right balance in the difficulty of the generated curriculum points. As a limitation, our method, which adds an additional DT-based curriculum generation to HER, obviously takes a longer time to compute than the original HER. However, as seen in our experiments, HERDT is more sample-efficient than HER. Another notable constraint of our implementation is that the DTs only accepts integer numbers. To address this limitation, we multiply floating-point numbers (position parameters) by a hyper-parameter and then truncate the fractional part. However, this truncation process reduces the algorithm's accuracy, as it treats positions with different fractional parts as the same position. To address this limitation and as part of future work, we will explore the use of multi-branch DTs as well as regression trees to handle continuous positions.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge (2018)
2. Mnih, V., et al.: Playing Atari with deep reinforcement learning. arXiv:1312.5602 (2013)
3. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
4. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
5. Rajeswaran, A., Lowrey, K., Todorov, E.V., Kakade, S.M.: Towards generalization and simplicity in continuous control. Adv. Neural Inf. Process. Syst. **30** (2017)
6. Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E.: Autonomous inverted helicopter flight via reinforcement learning. In: Ang, M.H., Khatib, O. (eds.) Experimental Robotics IX. STAR, vol. 21, pp. 363–372. Springer, Heidelberg (2006). https://doi.org/10.1007/11552246_35
7. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv:1509.02971 (2019)
8. Zakka, K., et al.: RoboPianist: a benchmark for high-dimensional robot control. arXiv:2304.04150 (2023)
9. Ng, A.Y., Harada, D., Russell, S.: Policy invariance under reward transformations: theory and application to reward shaping. In: International Conference on Machine Learning. (1999)
10. Rengarajan, D., Vaidya, G., Sarvesh, A., Kalathil, D., Shakkottai, S.: Reinforcement learning with sparse rewards using guidance from offline demonstration. In: International Conference on Learning Representations (2022)
11. Andrychowicz, M., et al.: Hindsight experience replay. Adv. Neural Inf. Process. Syst. **30** (2017)
12. Zhao, R., Tresp, V.: Energy-based hindsight experience prioritization. In: Conference on Robot Learning, pp. 113–122. PMLR (2018)
13. Zhao, R., Sun, X., Tresp, V.: Maximum entropy-regularized multi-goal reinforcement learning. In: International Conference on Machine Learning, pp. 7553–7562. PMLR (2019)
14. Puiutta, E., Veith, E.M.S.P.: Explainable reinforcement learning: a survey. In: Holzinger, A., Kieseberg, P., Tjoa, A.M., Weippl, E. (eds.) CD-MAKE 2020. LNCS, vol. 12279, pp. 77–95. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57321-8_5
15. Lipton, Z.C.: The mythos of model interpretability: in machine learning, the concept of interpretability is both important and slippery. Queue **16**(3), 31–57 (2018)
16. Molnar, C.: Interpretable machine learning. Lulu. com (2020)
17. Coppens, Y., et al.: Distilling deep reinforcement learning policies in soft decision trees. In: IJCAI Workshop on Explainable Artificial Intelligence, pp. 1–6 (2019)

18. Bastani, O., Pu, Y., Solar-Lezama, A.: Verifiable reinforcement learning via policy extraction. Adv. Neural Inf. Process. Syst. **31** (2018)
19. Ding, Z., Hernandez-Leal, P., Ding, G.W., Li, C., Huang, R.: CDT: cascading decision trees for explainable reinforcement learning. arXiv:2011.07553 (2020)
20. Roth, A.M., Topin, N., Jamshidi, P., Veloso, M.: Conservative Q-improvement: reinforcement learning for an interpretable decision-tree policy. arXiv:1907.01180 (2019)
21. Hallawa, A., et al.: Evo-RL: evolutionary-driven reinforcement learning. In: Genetic and Evolutionary Computation Conference Companion, pp. 153–154 (2021)
22. Custode, L.L., Iacca, G.: Evolutionary learning of interpretable decision trees. IEEE Access **11**, 6169–6184 (2023)
23. Ferigo, A., Custode, L.L., Iacca, G.: Quality diversity evolutionary learning of decision trees. In: Symposium on Applied Computing, pp. 425–432. ACM/SIGAPP (2023)
24. Custode, L.L., Iacca, G.: Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs. In: Genetic and Evolutionary Computation Conference Companion, pp. 224–227 (2022)
25. Custode, L.L., Iacca, G.: A co-evolutionary approach to interpretable reinforcement learning in environments with continuous action spaces. In: IEEE Symposium Series on Computational Intelligence, pp. 1–8. IEEE (2021)
26. Crespi, M., Ferigo, A., Custode, L.L., Iacca, G.: A population-based approach for multi-agent interpretable reinforcement learning. Appl. Soft Comput. **147**, 110758 (2023)
27. Todorov, E., Erez, T., Tassa, Y.: MuJoCo: A physics engine for model-based control. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5026–5033. IEEE (2012)
28. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**, 279–292 (1992)
29. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv:1509.02971 (2015)
30. Plappert, M., et al.: Multi-goal reinforcement learning: challenging robotics environments and request for research. arXiv:1802.09464 (2018)
31. Schaul, T., Horgan, D., Gregor, K., Silver, D.: Universal value function approximators. In: International Conference on Machine Learning. (2015)
32. Ryan, C., Collins, J.J., Neill, M.O.: Grammatical evolution: evolving programs for an arbitrary language. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, pp. 83–96. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055930
33. Brockman, G., et al.: OpenAI Gym. arXiv:1606.01540 (2016)