Evolutionary Reinforcement Learning with Parameterized Action Primitives for Diverse Manipulation Tasks

Xianxu Qiu[1], Haiming Huang[2], Weiwei Chen[2], Qiuzhen Lin[1*], Wei-Neng Chen[3], Fuchun Sun[4]

[1]College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
[2]College of Electronics and Information Engineering, Shenzhen University, Shenzhen, China
[3]School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
[4]Department of Computer Science and Technology, Tsinghua University, Beijing, China.
lijq@szu.edu.cn

Abstract

Reinforcement learning (RL) has shown promising performance in tackling robotic manipulation tasks (RMTs), which require learning a prolonged sequence of manipulation actions to control robots efficiently. However, most RL algorithms often suffer from two problems when solving RMTs: inefficient exploration due to the extremely large action space and catastrophic forgetting due to the poor sampling efficiency. To alleviate these problems, this paper introduces an Evolutionary Reinforcement Learning algorithm with parameterized Action Primitives, called ERLAP, which combines the advantages of an evolutionary algorithm (EA) and hierarchical RL (HRL) to solve diverse RMTs. A library of heterogeneous action primitives is constructed in HRL to enhance the exploration efficiency of robots and dual populations with new evolutionary operators are run in EA to optimize these primitive sequences, which can diversify the distribution of replay buffer and avoid catastrophic forgetting. The experiments show that ERLAP outperforms four state-of-the-art RL algorithms in simulated RMTs with dense rewards and can effectively avoid catastrophic forgetting in a set of more challenging simulated RMTs with sparse rewards.

Introduction

Reinforcement learning (RL) has shown promising performance in tackling robotic manipulation tasks (RMTs), which require learning a prolonged sequence of manipulation actions to control robots efficiently. However, most RL algorithms often suffer from two problems when solving RMTs: inefficient exploration due to the extremely large action space and catastrophic forgetting due to the poor sampling efficiency. To alleviate these problems, this paper introduces an Evolutionary Reinforcement Learning algorithm with parameterized Action Primitives, called ERLAP, which combines the advantages of an evolutionary algorithm (EA) and hierarchical RL (HRL) to solve diverse RMTs. A library of heterogeneous action primitives is constructed in HRL to enhance the exploration efficiency of robots and dual populations with new evolutionary operators are run in EA to optimize these primitive sequences, which can diversify the distribution of replay buffer and avoid catastrophic forgetting. The experiments show that ERLAP outperforms four state-of-the-art RL algorithms in simulated RMTs with dense rewards and can effectively avoid catastrophic forgetting in a set of more challenging simulated RMTs with sparse rewards.

In recent years, deep RL (DRL) has shown better learning ability to solve RMTs. Moreover, imitation learning (Li, Chappell, and Rojas 2023; Wang et al. 2022b; Zhang et al. 2024), curriculum learning (Yang et al. 2022; Li et al. 2021), and experience replay techniques (Schaul et al. 2015; Andrychowicz et al. 2017), are proposed for DRL to better solve RMTs. However, these DRL algorithms have to explore the robotic state space by raw actions (e.g., force and torque) or joint skill, which is a prohibitively large action space for DRL to complete a specific task. Thus, some recent DRL algorithms (Fu et al. 2020; Singh et al. 2020; Mandlekar et al. 2020) adopt offline data to speed up the learning, which can alleviate the exploration burden, but they are time-consuming to scale up for a new RMT as they need to collect offline data each time.

On the other hand, hierarchical RL (HRL) is also designed to alleviate the exploration burden, which decomposes the target RMT into sub-tasks and addresses them separately. HRL with parameterized action primitives (Lee, Yang, and Lim 2019; Chitnis et al. 2020; Nasiriany, Liu, and Zhu 2022; Wang et al. 2023; Dalal, Pathak, and Salakhutdinov 2021) is a promising paradigm for robotic skill learning, which explores the robotic state space by primitives to alleviate the exploration burden. Specifically, these algorithms determine the primitive type at a high-level policy and corresponding parameters at a low-level policy. Although these HRL algorithms have partially alleviated the exploration burden, they still highly rely on informative rewards and suffer from catastrophic forgetting, preventing them from scaling up to more complex RMTs.

Moreover, evolutionary RL (ERL) (Khadka and Tumer 2018) has recently been proposed for overcoming the catastrophic forgetting, which combines the advantages of evolutionary algorithms (EAs) (Spears et al. 1993) and DRL. Some ERL algorithms (Jianye et al. 2022; Khadka and Tumer 2018; Bodnar, Day, and Lió 2020) utilize a population to store network parameters, which are evolved to alleviate the parameters forgetting problem during training. Be-

*Corresponding author.

sides, other ERL algorithms (Ma et al. 2022) exploit EAs to evolve promising actions, which can diversify the distribution of replay buffer and alleviate the action forgetting problem. However, these ERL algorithms are prone to network collapse or lack semantics, which also have a large exploration space in solving RMTs.

In summary, a number of DRL algorithms have been proposed to address RMTs, which have their specific advantages: HRL algorithms are able to alleviate the exploration burden while ERL algorithms can better overcome catastrophic forgetting. However, to the best of our knowledge, no study has combined HRL and ERL to address the above two challenges (inefficient exploration and catastrophic forgetting) simultaneously. To fill this research gap, this article proposes an **E**volutionary **R**einforcement **L**earning algorithm with parameterized **A**ction **P**rimitives (ERLAP), which integrates EA and HRL with predefined action primitives to better solve RMTs. A parameterized action space of HRL is adopted in ERLAP to replace the raw action space, and an EA population is run to store and evolve high-quality primitive sequences. In this way, ERLAP shows more promising performance in addressing RMTs. The main contributions of this work are summarized as follows:

- A library of parameterized action primitives is constructed in HRL to replace raw actions, which enhances the exploration efficiency and avoids complex raw action planning for solving RMTs.
- A dual-population evolutionary mechanism with new evolutionary operators is designed to store and evolve high-quality primitive sequences, which avoids catastrophic forgetting during training.
- A new ERL framework called ERLAP is proposed by combining the above library of primitives and dual-population evolutionary mechanism. The experiments demonstrate that ERLAP outperforms four state-of-the-art RL algorithms in six RMTs and also effectively avoids catastrophic forgetting in a set of more challenging RMTs.

Related Work and Motivation

This section gives a brief review of HRL and ERL for addressing RMTs, and then describes our motivation.

HRL for Solving RMTs. In recent decade, a number of HRL algorithms (Liu et al. 2022) have been proposed to improve exploration efficiency in solving RMTs. Generally, the HRL algorithms can be classified into two main kinds, i.e., feudal-based and options-based HRL algorithms.

The feudal-based HRL algorithms learn a high-level policy to generate intermediate sub-goals achieved by low-level policies, which focus on task decomposition. For example, (Xia et al. 2021) proposed the ReLMoGen framework, in which the high-level policy is utilized to predict sub-goals and the low-level policy as a classical motion generator is responsible for planning and executing the motion to achieve these sub-goals. (Yun et al. 2022) proposed the Gaussian Random Trajectory guided Hierarchical reinforcement Learning (GRT-HL) method, where a probabilistic latent variable model is learned as the high-level policy to construct continuous latent representations of trajectories and the low-level policy controls the end-effector to reach these sub-goals of reference trajectory orderly. In contrast to GRT-HL, (Wang et al. 2022a) proposed the Hierarchical Decoupling Optimization (HDO) framework, in which traditional Rapidly-exploring Random Trees (RRT) are used to complete trajectory planning and the low-level policy is trained via DDPG to achieve the position and orientation of planned trajectory. (Zhang et al. 2022) transformed the raw action space into a number of adjacent action spaces for high-level planning, which reduces the exploration burden and improves the sample efficiency.

The options-based HRL algorithms learn a high-level policy to choose low-level options or skills, which focus on the combination of options. For example, (Strudel et al. 2020) proposed a task planning approach to combine primitive skills, which are pre-trained by expert dataset. (Sharma et al. 2020) used a neural network to dynamically compose sub-tasks, which can be achieved by task-axis controllers. (Lee et al. 2018) proposed a transition policies (TP) method, which effectively connects primitive skills to solve the sequential tasks without handcrafted rewards. Furthermore, a parameterized primitive can be viewed as a special option that needed to be instantiated by a low-level network. (Chitnis et al. 2020) trained a state-independent model to predict primitive type and a state-dependent neural network to predict its parameters. (Nasiriany, Liu, and Zhu 2022) and (Dalal, Pathak, and Salakhutdinov 2021) promoted the exploration efficiency of standard DRL methods with a set of pre-defined primitives, which train two state-dependent networks to learn primitive type and its parameters, respectively. (Wang et al. 2023) proposed the TRAPs method, which embeds a linear temporal logic (LTL) into (Nasiriany, Liu, and Zhu 2022) and decomposes the training task at an abstract level to facilitate high-level policy learning.

ERL for Solving RMTs. A number of ERL algorithms have been designed to overcome the catastrophic forgetting during training, which are mostly designed based on two main frameworks: action evolution and policy evolution.

The action evolution framework utilizes EAs to search actions with higher fitness, optimizing the experience distribution of replay buffer. For example, (Ma et al. 2022) proposed the Evolutionary Action Selection-Twin Delayed Deep Deterministic Policy Gradient (EAS-TD3), which uses Particle Swarm Optimization (PSO) to optimize the action selection of RL agents. The QT-Opt algorithm (Kalashnikov et al. 2018) leveraged cross-entropy method (CEM) to optimize the action selected from a batch of randomly sampled actions. GRAC (Shao et al. 2022) searched better actions associated with higher Q-values in a broad neighborhood, which guides the update of the actor network.

The policy evolution framework periodically inserts an RL agent into the EA population, which injects gradient information into the EA. Then, this population is evolved to diversify experiences of replay buffer, which contributes to the training of the RL agent. This evolution framework was first proposed by (Khadka and Tumer 2018) as a promising paradigm to avoid catastrophic forgetting. Recently, a num-

ber of researchers have improved this framework. For example, PDERL (Bodnar, Day, and Lió 2020) utilized Q-filtered distillation crossover and proximal mutation operators to alleviate policy crash. ERL-Re^2 (Jianye et al. 2022) designed a behavior level evolution operator and a linear policy representation to improve the sampling efficiency of fitness estimation. CERL (Khadka et al. 2019) extended a single actor-critic agent to multiple agents with different hyperparameters and used UCB to select training agents, which prevents high sensitivity to hyperparameters. CEM-RL (Pourchot and Sigaud 2018) combined CEM and the Twin Delayed Deep Deterministic policy gradient (TD3) for policy search, which offers a satisfactory trade-off between sampling efficiency and performance.

Motivation. Although a number of HRL and ERL algorithms have been proposed above, few studies have combined HRL and ERL to address RMTs, as neither action evolution nor policy evolution can reduce the search space and capture semantic relationships between actions. Action evolution focuses only on a single action, but there is a significant precedence relationship between actions in RMTs, while policy evolution focuses on network parameters or neural architectures, which is inapplicable to HRL and easy to cause network collapse. Thus, to the best of our knowledge, the proposed ERLAP is the first try to integrate HRL and ERL for tackling RMTs, which leverages parameterized primitives to replace raw actions and directly stores entire primitive sequences for evolution. By this way, ERLAP alleviates the exploration burden and avoids catastrophic forgetting in solving RMTs.

The Proposed ERLAP

As shown in Figure 1, the architecture of ERLAP consists of two main parts: an RL module and an EA module. The RL module uses HRL to generate a sequence of primitives, including the primitive types and their corresponding parameters. The EA module uses populations to store high-quality sequences from the RL module and then evolves these sequences by the proposed evolutionary operators. After that, these sequences will interact with the environment again. With the help of the experience gained from the EA module, the RL agent can effectively address RMTs. The specific details of ERLAP are presented below.

Parameterized Primitive Building for ERLAP

Generally, a finite number of primitives are executed to solve an RMT and a library of parameterized primitives is used in ERLAP. Here, an execution length l_{type} is used to represent the complexity of executing primitives. When the cumulative execution length of a primitive sequence exceeds the threshold L, i.e., $\sum_i l_{type} > L$, the RL module stops generating new primitives. The details of each primitive are described as follows:

Reach: The robot moves its end-effector to a target location (x, y, z) at a yaw angle ψ. 4-D parameters are needed and l_{reach} is fixed at 20.

Release: The robot moves its end-effector to a target location (x, y, z) at a yaw angle ψ, and opens its gripper. 4-D parameters are needed and $l_{release}$ is fixed at 20.

Open: The robot opens its gripper at a yaw angle ψ. 1-D parameter is needed and l_{open} is fixed at 1.

Grasp: The robot moves its end-effector to a target location (x, y, z) at a yaw angle ψ, and closes its gripper. 4-D parameters are needed and l_{grasp} is fixed at 20.

Micro-move: The robot moves its end-effector by a micro displacement $(\delta_x, \delta_y, \delta_z)$, and an angular offset δ_ψ. The micro displacement range is (-1 mm, 1 mm), and the angular offset range is (-2, 2) degree. 4-D parameters are needed and l_{micro} is fixed at 2.

All of these primitives can be viewed as functional $APIs$

Algorithm 1: ERLAP

Initialize Q network $Q_\theta(s, \xi)$, task policy $\pi_a(a|s)$ and parameter networks $\pi_p^a(x|s)$ with weights θ, ϕ and ψ
Initialize an empty cyclic replay buffer $\mathcal{D}$
for iteration $1 : M$ **do**
  Initialize $t \leftarrow 0$, $f \leftarrow 0$ and an empty set S_{RL}
  while episode not terminated **do**
    Sample primitive type a_t from $\pi_a(a_t|s_t)$
    Sample primitive parameters x_t from $\pi_p^{a_t}(x_t|s_t)$
    Combine a_t and x_t as ξ_t
    r_t, s_{t+1} = Execute (ξ_t,s_t)
    Append transition $(s_t, \xi_t, r_t, s_{t+1})$ to $\mathcal{D}$
    Update $S_{RL} \leftarrow (\xi_0, ..., \xi_t)$
    Update fitness and timer $f \leftarrow f + r_t$, $t \leftarrow t + 1$
  end while
  Store S_{RL} according to f
  $\mathcal{D} \leftarrow$ evolve the sequences each T time
  for training step $1, ..., K$ **do**
    Update θ, ϕ and ψ by Eqs. (2), (3) and (4), respectively
  end for
end for

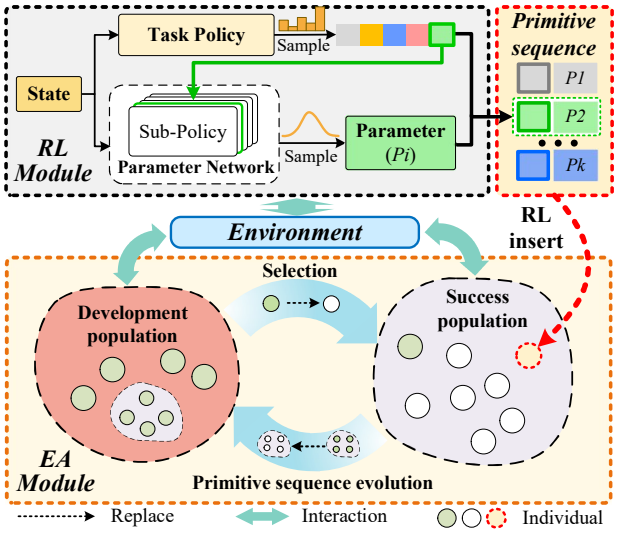


Figure 1: The architecture of ERLAP with two main parts: an RL module and an EA module.

that execute the corresponding action based on the input parameters. The environment translates these parameterized primitives into a series of low-level robotic motions to execute, reducing the ineffective exploration by the RL agent.

Primitive Sequence Generation via RL Module

After building the library of parameterized primitives, the primitive sequence generation via the RL module is introduced in this subsection. In general, the decision-making problem of RL is modelled as a Markov decision process (MDP), defined by the tuple $M = (S, A, r, p, p_0, \gamma)$, where S is the state space, A is the action space, r is the reward function, p is the transition function, p_0 is the initial state distribution, and $\gamma \in [0, 1)$ is the discount factor. The objective of RL is to maximize the accumulated reward ($R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$) over all episodes. ERLAP considers an MDP with a continuous state space and a discrete parameterized action space A, as follows:

$$A = \bigcup_{a \in A_d} \{(a, x) | x \in X_a\} \tag{1}$$

where A_d represents a set of parameterized action primitives type, and each $a \in A_d$ has a corresponding parameter $x \in R^{d_a}$ (d_a is the dimension of X_a). Each action a and its corresponding parameter x form an action primitive $\xi = (a, x)$ (introduced in the previous section). In ERLAP, the RL agent executes a ξ instead of a raw action at each decision-making step. Therefore, the RL module can be divided into two phases: exploration phase and learning phase.

The details of the exploration phase are found in lines 5-13 of Algorithm 1. The high-level task network π_a is a single neural network used to select the primitive for execution, while the low-level parameter network $\pi_p^a = \{\pi_p^{a_1}, ..., \pi_p^{a_n}\}$ is a collection of sub-networks used to instantiate a primitive. In line 6, the task network $\pi_a(a_t|s_t)$ takes the current robot state s_t as input and outputs a primitive type a_t. Then, in line 7, the corresponding parameter network $\pi_p^{a_t}(x_t|s_t)$ is chosen to instantiate a_t by the output parameter x_t. In lines 8-9, these primitives will be executed to generate experience and stored in the replay buffer for training. Finally, in lines 11-12, the primitive sequence is stored in S_{RL} for the EA module.

The learning phase focuses on how interaction experiences can be used to update networks. In ERLAP, a state-of-the-art DRL algorithm SAC (Haarnoja et al. 2018), is chosen for the RL module. The losses are designed as follows:

$$\begin{aligned} J_Q(\theta) = (Q_\theta(s, \xi) - (r(s, \xi) + \gamma(Q_{\theta'}(s', \xi') - \\ \alpha_t log(\pi_a(a'|s')) - \alpha_p log(\pi_p^{a'}(x'|s'))))^2 \end{aligned} \tag{2}$$

$$J_{\pi_a}(\phi) = E_{a \sim \pi_a}[\alpha_t log(\pi_a(a|s)) - E_{x \sim \pi_p} Q_\theta(s, \xi)] \tag{3}$$

$$J_{\pi_p}(\psi) = E_{a \sim \pi_a} E_{x \sim \pi_p}[\alpha_p log(\pi_p^a(x|s)) - Q_\theta(s, \xi)] \tag{4}$$

where the critic neural network $Q_\theta(s, \xi)$ is used to evaluate the primitive quality. Currently, α_t and α_p represent the maximum entropy targets for the task network and parameter network, respectively.

Dual-population Evolutionary Mechanism

To avoid catastrophic forgetting caused by the inability to sample high-quality experience from the RL module, ERLAP addresses this problem by storing and evolving these experiences through EA. Specifically, a complete primitive sequence is treated as an individual in the EA, and the fitness of each individual is calculated as the accumulated reward, which is applied to evaluate the quality of the individual. In the EA module, two populations are designed: the success population (P_s) and the development population (P_d). P_s is responsible for storing high-quality individuals from the RL module and providing diverse individuals for P_d, while P_d is responsible for selecting high-quality individuals to optimize P_s.

Specifically, if the weakest individual in P_s is worse than the sequence S_{RL} from the RL module, it is replaced by S_{RL}, which can retain the high-quality experience. Then, for each of T epochs, the individuals in P_s are probabilistically perturbed by the primitive sequence evolutionary operator (introduced in next section), and the resultant offspring are stored in P_d. Subsequently, the individuals in P_s and P_d will interact with the environment and update their fitness values.

In P_s, the individuals are selected by random sampling and their fitness values f_i are updated according to the weight factor, as follows:

$$f_i \leftarrow \alpha * f_i + \beta * f \tag{5}$$

where α is the old fitness weight, β is the new fitness weight, and f denotes the new fitness value obtained from the interaction. Here, α is fixed as 0.9 and β is fixed as 0.1.

As the number of individuals in P_d is relatively large than that in P_s, the sampling operation in P_d requires better trade-off between exploitation and exploration. Therefore, an upper confidence bound (UCB) score $\mathcal{U}$ is computed for each individual in P_d to determine which individual should be sampled, as follows:

$$\mathcal{U}_i = \begin{cases} \infty, & \text{if } n_i == 0 \\ f_i + C * \sqrt{log(N)/n_i}, & \text{others} \end{cases} \tag{6}$$

where n_i is the sampling time for a particular individual in P_d, N is the sum of the sampling times in P_d, and C is a hyper parameter fixed at 4 in ERLAP. After interacting with the environment, the fitness value f_i is updated as follows:

$$f_i \leftarrow \frac{1}{n_i}((n_i - 1) * f_i + f) \tag{7}$$

After updating the fitness, if the best individual in P_d is better than the weakest individuals in P_s, it will replace the weakest one in P_s, ensuring the convergence of P_s.

Sequence Evolution in EA Module

In ERLAP, the primitive sequence evolution is realized by one crossover and four mutation operators. As shown in Figure 2, some individuals are selected from P_s as parents, which are then evolved in the following ways:

Crossover: As shown in Figure 2(a), the top 2 individuals in P_s are selected as parents. Then, two segments randomly selected from the parents are swapped. The length of these

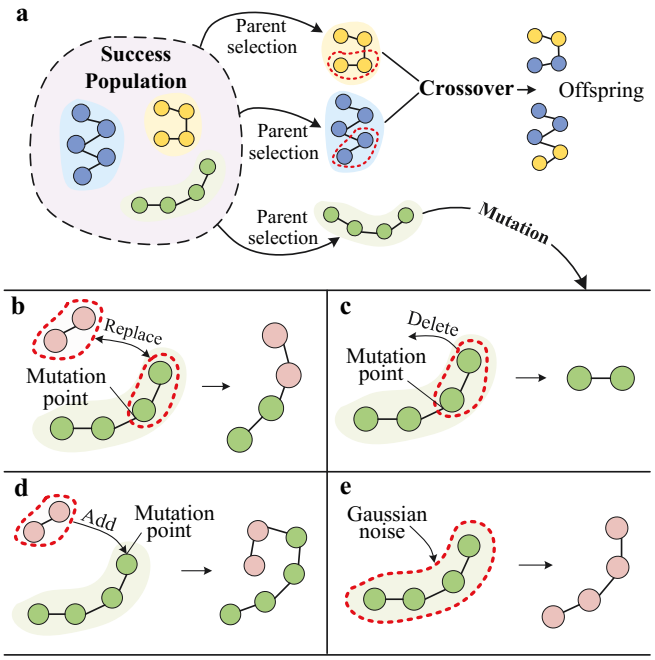


Figure 2: The process of primitive sequence evolutionary operation, including (a) Crossover, (b) Replacement mutation, (c) Deletion mutation, (d) Addition mutation, and (e) Gaussian disturbance mutation.

segments is limited to between 1/4 and 1/2 of its parent's maximum length. **Mutation**: Each individual selected from P_s undergoes a mutation operation. The mutation operators in ERLAP include four specific types: replacement, deletion, addition and Gaussian disturbance mutations. In the replacement mutation, the parent segment is replaced by a randomly generated segment. In the deletion mutation, the selected segment is randomly removed from the parent. In the addition mutation, a new segment is inserted into the parent. Finally, the Gaussian disturbance mutation adds Gaussian noise to each parameter of the entire primitive sequence without changing its basic order.

As the replacement, deletion and addition mutations change the primitive distribution (e.g., the primitive type and primitive order) of the parent, their parameters can not be directly used to address the training task. Therefore, these offspring must be re-entered into the parameter network before interacting with the environment. In contrast, after the Gaussian disturbance mutation, the offspring can directly interact with the environment because they only change the primitive parameters rather than the primitive distribution.

Experiments

In this section, the performance of ERLAP is investigated in detail. The experimental setup consists of two parts. Firstly, we verify the sampling efficiency of ERLAP in Robosuite (Zhu et al. 2020) environment with dense reward, and then verify that ERLAP can effectively solve catastrophic forgetting in more challenging RMTs with sparse reward.

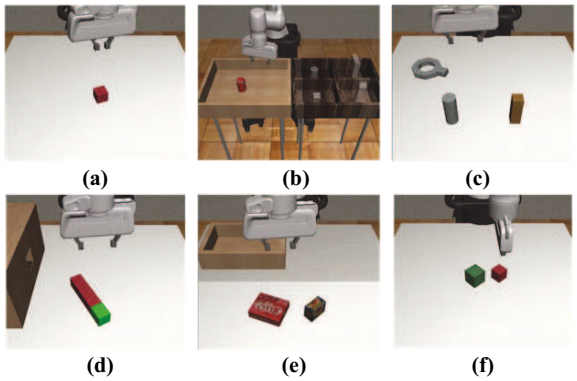


Figure 3: Simulated environments on Robosuite with dense reward, including: (a) `Lift` (`LIF`), (b) `Pick-and-Place` (`PAP`), (c) `Nut-Assembly` (`NAS`), (d) `Peg-Insert` (`PIN`), (e) `Clean-Up` (`CUP`) and (f) `Stack` (`STA`).

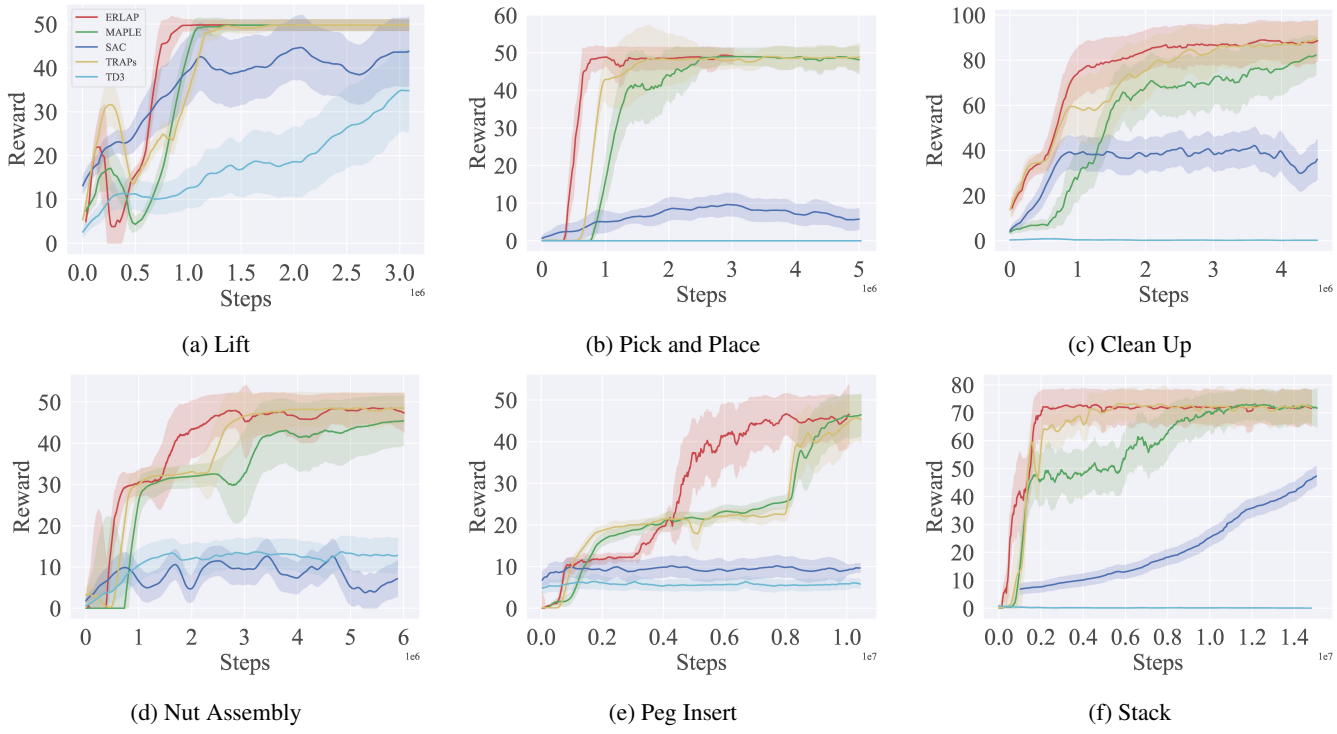
Performance Metric under Dense Rewards

Experimental Setup and Baselines. In this experiment, Robosuite, a simulated environment for RMTs, is chosen to evaluate the sampling performance of ERLAP. As shown in Figure 3, six manipulation tasks of various complexities are selected. ERLAP is compared with some state-of-the-art RL algorithms, including: 1) **TD3** (Fujimoto, Hoof, and Meger 2018): the baseline algorithm in Robosuite; 2) **SAC** (Haarnoja et al. 2018): the state-of-the-art DRL algorithm trained on end-effector commands; 3) **MAPLE** (Nasiriany, Liu, and Zhu 2022): an augmented RL framework built upon a library of predefined behavioral primitives; 4) **TRAPs** (Wang et al. 2023): an HRL-based framework to combine the LTL and parameterized behavior primitives, which is the best manipulation skill learning framework currently. It is worth noting that due to the introduction of the dual-population mechanism, the experience of ERLAP comes from two parts: population interaction and network exploration. To be fair, we set the sampling steps from network of ERLAP to be half of the above baseline.

Experimental Results and Discussion. Figure 4 shows the reward values obtained by all the compared algorithms during training, which are averaged over three random seeds. It can be observed that HRL algorithms with parameterized action primitives (MAPLE, TRAPs, and ERLAP) significantly outperform those without parameterized action primitives (TD3 and SAC). Especially when complex skills (`PAP`, `NAS`, `PIN`, and `STA`) are considered, TD3 and SAC are difficult to achieve high rewards or even get 0 reward due to large exploration space.

ERLAP uses fewer steps compared to all baselines, with a reduction of more than 18.8% against TRAPs and a reduction of 30.1% against MAPLE. This indicates that the dual-population evolution mechanism can improve the experience distribution in replay buffer and the sampling quality.

We evaluate the training policy 100 times with the mean of their success rates as our final task success rate, which is listed in Table 1. It can be observed that although ERLAP



(a) Lift (b) Pick and Place (c) Clean Up

(d) Nut Assembly (e) Peg Insert (f) Stack

Figure 4: The results of reward curves averaged over 3 seeds with shaded regions depicting the standard deviation.

Algorithms	LIF	PAP	NAS	PIN	CUP	STA	**Average Success Rate**
TD3	75.0±12.7	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	12.50
SAC	98.0±2.4	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	28.0±28.7	21.00
MAPLE	100.0±0.0	95.0±7.7	99.0±2.0	100.0±0.0	91.5±5.8	98.0±2.4	97.25
TRAPs	100.0±0.0	99.3±2.6	100.0±0.0	98.5±3.6	95.0±2.1	98.5±2.5	98.55
ERLAP	100.0±0.0	98.12±7.4	100.0±0.0	100.0±0.0	97.5±3.2	100.0±0.0	**99.27**

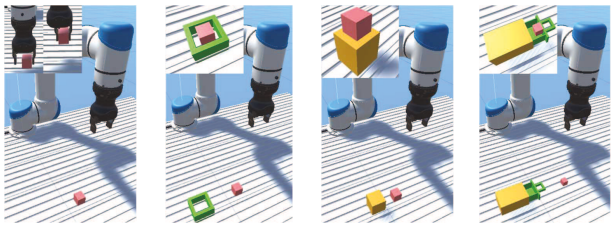
Table 1: The result of final task success rates (%) and standard deviation of each RMT with dense rewards

achieves a similar success rate with TRAPs and MAPLE, ERLAP slightly outperforms them in terms of average success rate. Furthermore, ERLAP achieved a 100% success rate in four RMTs, while MAPLE and TRAPs only solved two RMTs with 100% success rate.

In conclusion, these results validate that ERLAP can solve a wide range of RMTs and outperform all baselines, due to the combination of HRL and EA for more efficient sampling.

Performance Metric under Sparse Rewards

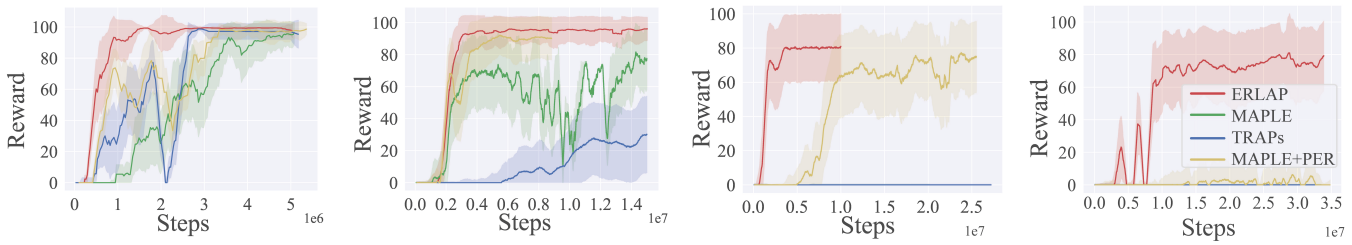
Experimental Setup and Baselines. This experiment is conducted to verify whether ERLAP can effectively prevent catastrophic forgetting in RMTs with sparse rewards. As shown in Figure 5, four tasks with sparse rewards are designed. In this experiment, ERLAP is compared with two primitive-based algorithms **MAPLE** and **TRAPs**. An experience replay technique (PER) (Schaul et al. 2015), which solves the forgetting problem, is inserted in MAPLE (**MAPLE+PER**) as another baseline, with half of the experience coming from successful experiences and the other half from failed experiences.



(a) (b) (c) (d)

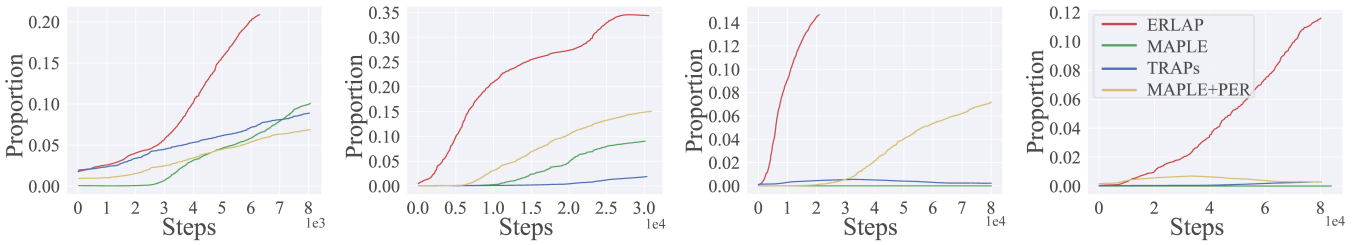
Figure 5: Simulated environments on Unity with sparse reward, from left to right, are LIF, PAP, STA, and Put-into-Drawer (PID). The ending state of a successfully completed episode is shown in the nested image in the upper right corner of each plot.

Experimental Results and Discussion. We evaluate the training policy 100 times with the mean of their success rates as our final success rate, which is listed in Table 2. The reward curves for the tasks during training are shown in Figure 6. In addition, we visualize the experience proportion of the



(a) Lift (b) Pick and Place (c) Stack (d) Put into Drawer

Figure 6: The results of reward curves of each RMT with sparse rewards. All rewards are normalized to 0-100.



(a) Lift (b) Pick and Place (c) Stack (d) Put into Drawer

Figure 7: The experience distribution change curve in replay buffer of each RMT with sparse reward

Task	ERLAP	MAPLE	TRAPs	MAPLE+PER
LIF	**100.0±0**	99.1±2.5	98.5±2.7	99.2±3.1
PAP	**98.5±3.1**	78.3±20.6	30.5±27.0	92.3±9.1
STA	**81.8±20.0**	0.0±0.0	0.0±0.0	74.6±16.7
PID	**77.6±11.5**	0.0±0.0	0.0±0.0	0.0±0.0

Table 2: The result of final task success rates (%) and standard deviation of each RMT with sparse rewards

replay buffer in Figure 7, which is calculated as follows:

$$Proportion = \frac{number\ of\ successful\ experience}{number\ of\ failed\ experience} \quad (8)$$

It can be observed that 1) the LIF task can be solved by both algorithms , but ERLAP achieves a higher success rate of 100%. Furthermore, ERLAP can solve this problem using fewer steps, with over 66.6% reduction compared to MAPLE, 40% reduction compared to TRAPs, and 53.1% reduction compared to MAPLE+PER.

2) ERLAP, MAPLE+PER, MAPLE and TRAPs achieve success rates of 98.5%, 92.3%, 78.3% and 30.5% in the PAP task, respectively. As shown in Figure 7(b), the experience distribution of ERLAP and MAPLE+PER are more uniform than other algorithms, showing better performance. This also indicates that when the experience distribution in the replay buffer is not uniform (similar to the long tail effect), random sampling is difficult to sample high-quality experiences, which further leads to poor training results.

3) ERLAP outperforms other algorithms in the STA and PID tasks. In the STA task, ERLAP achieves a success rate of 81.8%, higher than 74.6% obtained by MAPLE+PER, and ERLAP has 41.7% fewer sampling steps than MAPLE. In the PID task, ERLAP achieves a success rate of 77.6%, while other algorithms are difficult to get any reward due to its inability to store and reuse successful experiences.

4) As shown in Figure 7, the failure of other algorithms in these tasks does not stem from a lack of exploration into successful experiences, for example, successful experiences can be sampled from TRAPs in the STA task and MAPLE+PER in the PID task. However, ERLAP utilizes the EA populations to store the successful experiences during training, and reuses these high-quality experiences through primitive evolution operators, which can optimize the distribution of data in the replay buffer and avoid catastrophic forgetting.

5) As shown in Figures 6 and 7, the sample distribution of the replay buffer directly affects the final training result. Figure 7 shows that the proportion change of ERLAP significantly outperforms other algorithms, because the dual-population evolution mechanism of ERLAP can not only store experience like PER, but also continue to derive and utilize high-quality experience, which effectively solves the catastrophic forgetting problem under sparse reward.

Conclusion

This paper has proposed a general skill learning framework (ERLAP), which is the first try to combine EA and HRL with parameterized action primitives to solve various RMTs. ERLAP outperforms other RL algorithms on six RMTs with dense rewards and avoids catastrophic forgetting on challenging RMTs with sparse rewards. An important avenue of our future work is to automatically discover primitives and add them to the primitive library. Moreover, using the large language model to leverage the knowledge graph inference primitive sequence is also an interesting direction.

Acknowledgments

This work was supported by the Major Project of the New Generation of Artificial Intelligence (No. 2018AAA0102900), the National Natural Science Foundation of China (No. 62376163 and No. 62173233), the Guangdong Regional Joint Foundation Key Project under Grant 2022B1515120076, the Shenzhen Science and Technology Innovation Commission project (JCYJ20220531102809022 and JCYJ20220531101411027), the Scientific Instrument Developing Project of ShenZhen University (2023YQ021).

References

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.

Bodnar, C.; Day, B.; and Lió, P. 2020. Proximal distilled evolutionary reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 3283–3290.

Chitnis, R.; Tulsiani, S.; Gupta, S.; and Gupta, A. 2020. Efficient bimanual manipulation using learned task schemas. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 1149–1155. IEEE.

Dalal, M.; Pathak, D.; and Salakhutdinov, R. R. 2021. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34: 21847–21859.

Fu, J.; Kumar, A.; Nachum, O.; Tucker, G.; and Levine, S. 2020. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

Fujimoto, S.; Hoof, H.; and Meger, D. 2018. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, 1587–1596. PMLR.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. PMLR.

Jianye, H.; Li, P.; Tang, H.; Zheng, Y.; Fu, X.; and Meng, Z. 2022. ERL-Re ?2: Efficient Evolutionary Reinforcement Learning with Shared State Representation and Individual Policy Representation. In *The Eleventh International Conference on Learning Representations*.

Kalashnikov, D.; Irpan, A.; Pastor, P.; Ibarz, J.; Herzog, A.; Jang, E.; Quillen, D.; Holly, E.; Kalakrishnan, M.; Vanhoucke, V.; et al. 2018. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*.

Khadka, S.; Majumdar, S.; Nassar, T.; Dwiel, Z.; Tumer, E.; Miret, S.; Liu, Y.; and Tumer, K. 2019. Collaborative evolutionary reinforcement learning. In *International conference on machine learning*, 3341–3350. PMLR.

Khadka, S.; and Tumer, K. 2018. Evolution-guided policy gradient in reinforcement learning. *Advances in Neural Information Processing Systems*, 31.

Lee, Y.; Sun, S.-H.; Somasundaram, S.; Hu, E. S.; and Lim, J. J. 2018. Composing complex skills by learning transition policies. In *International Conference on Learning Representations*.

Lee, Y.; Yang, J.; and Lim, J. J. 2019. Learning to coordinate manipulation skills via skill behavior diversification. In *International conference on learning representations*.

Li, K.; Chappell, D.; and Rojas, N. 2023. Immersive Demonstrations are the Key to Imitation Learning. *arXiv preprint arXiv:2301.09157*.

Li, Y.; Kong, T.; Li, L.; Li, Y.; and Wu, Y. 2021. Learning to design and construct bridge without blueprint. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2398–2405. IEEE.

Liu, S.; Tang, T.; Zhang, W.; Cui, J.; and Xu, X. 2022. A Brief Review of Recent Hierarchical Reinforcement Learning for Robotic Manipulation. In *2022 10th International Conference on Information Systems and Computing Technology (ISCTech)*, 721–728. IEEE.

Ma, Y.; Liu, T.; Wei, B.; Liu, Y.; Xu, K.; and Li, W. 2022. Evolutionary Action Selection for Gradient-Based Policy Learning. In *International Conference on Neural Information Processing*, 579–590. Springer.

Mandlekar, A.; Ramos, F.; Boots, B.; Savarese, S.; Fei-Fei, L.; Garg, A.; and Fox, D. 2020. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 4414–4420. IEEE.

Nasiriany, S.; Liu, H.; and Zhu, Y. 2022. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. In *2022 International Conference on Robotics and Automation (ICRA)*, 7477–7484. IEEE.

Pourchot, A.; and Sigaud, O. 2018. CEM-RL: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222*.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Shao, L.; You, Y.; Yan, M.; Yuan, S.; Sun, Q.; and Bohg, J. 2022. Grac: Self-guided and self-regularized actor-critic. In *Conference on Robot Learning*, 267–276. PMLR.

Sharma, M.; Liang, J.; Zhao, J.; LaGrassa, A.; and Kroemer, O. 2020. Learning to compose hierarchical object-centric controllers for robotic manipulation. *arXiv preprint arXiv:2011.04627*.

Singh, A.; Liu, H.; Zhou, G.; Yu, A.; Rhinehart, N.; and Levine, S. 2020. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*.

Spears, W. M.; De Jong, K. A.; Bäck, T.; Fogel, D. B.; and De Garis, H. 1993. An overview of evolutionary computation. In *European conference on machine learning*, 442–459. Springer.

Strudel, R.; Pashevich, A.; Kalevatykh, I.; Laptev, I.; Sivic, J.; and Schmid, C. 2020. Learning to combine primitive skills: A step towards versatile robotic manipulation §. In

2020 IEEE International Conference on Robotics and Automation (ICRA), 4637–4643. IEEE.

Wang, H.; Zhang, H.; Li, L.; Kan, Z.; and Song, Y. 2023. Task-Driven Reinforcement Learning With Action Primitives for Long-Horizon Manipulation Skills. *IEEE Transactions on Cybernetics*, 1–14.

Wang, S.; Cao, Y.; Zheng, X.; and Zhang, T. 2022a. Collision-free trajectory planning for a 6-DoF free-floating space robot via hierarchical decoupling optimization. *IEEE Robotics and Automation Letters*, 7(2): 4953–4960.

Wang, Y.; Beltran-Hernandez, C. C.; Wan, W.; and Harada, K. 2022b. An adaptive imitation learning framework for robotic complex contact-rich insertion tasks. *Frontiers in Robotics and AI*, 8: 777363.

Xia, F.; Li, C.; Martín-Martín, R.; Litany, O.; Toshev, A.; and Savarese, S. 2021. Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 4583–4590. IEEE.

Yang, X.; Ji, Z.; Wu, J.; and Lai, Y.-K. 2022. Abstract demonstrations and adaptive exploration for efficient and stable multi-step sparse reward reinforcement learning. In *2022 27th International Conference on Automation and Computing (ICAC)*, 1–6. IEEE.

Yun, W. J.; Mohaisen, D.; Jung, S.; Kim, J.-K.; and Kim, J. 2022. Hierarchical reinforcement learning using Gaussian random trajectory generation in autonomous furniture assembly. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 3624–3633.

Zhang, D.; Fan, W.; Lloyd, J.; Yang, C.; and Lepora, N. F. 2024. One-Shot Domain-Adaptive Imitation Learning via Progressive Learning Applied to Robotic Pouring. *IEEE Transactions on Automation Science and Engineering*, 21(1): 541–554.

Zhang, T.; Guo, S.; Tan, T.; Hu, X.; and Chen, F. 2022. Adjacency constraint for efficient hierarchical reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4): 4152–4166.

Zhu, Y.; Wong, J.; Mandlekar, A.; Martín-Martín, R.; Joshi, A.; Nasiriany, S.; and Zhu, Y. 2020. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*.