Heuristically Adaptive Policy Reuse in Reinforcement Learning

Han-Chao Wang, Tianpei Yang, Jianye Hao*
College of Intelligence and Computing, Tianjin University
Tianjin, China
ares1899@126.com,tpyang@tju.edu.cn,jianye.hao@tju.edu.cn

ABSTRACT

Transfer learning can significantly improve the reinforcement learning process by leveraging prior knowledge from past learned tasks. However, how to select an optimal source task for reuse to improve a reinforcement learning agent is still challenging. In this paper, we propose a new Policy Reuse framework called Heuristically Adaptive Policy Reuse (HAPR) that facilitates efficient reuse of source policies, which is stored in a given Policy Library, by rapidly selecting the most appropriate policy only with its useful part. For the agent to reuse in successive tasks, HAPR is also capable of rebuilding the Policy Library to provide representative policies, whose quality is guaranteed by using KL-divergence to measure the irrelevance between policies. Our extensive experiments based on a grid-world domain show the efficiency and robustness of our method, compared with the state-of-the-art policy reuse approaches.

KEYWORDS

Policy Reuse; Transfer Learning; Reinforcement Learning

1 INTRODUCTION

Reinforcement learning (RL) [19] is a proverbial framework for an agent to learn and optimize a policy through the interactions with the environment and feedback rewards. Most RL methods are faced with sample efficiency problems, which makes it difficult to learn from scratch, especially in solving complex tasks. Transfer learning (TL) [20] can facilitate RL to improve learning efficiency by transferring past learned knowledge to a new task, which usually consists of three phases: select useful knowledge to transfer, find a suitable way to transfer and last, execute the transfer process. If the transferred knowledge which is incompatible with the target environment is selected, negative transfer occurs [12]. It is challenging to predict whether source information is useful in advance.

Policy Reuse is useful to accelerate RL process by reusing past similar learned policies. Existing researches on policy reuse include reusing expert suggestions [3, 18], defining the policy similarities in a reward shaping manner [2], modeling policy selection as Bayesian optimization problems [13], and transferring the experience instances of a source task to the target task by reusing these instances to estimate the reward function [9]. However, these approaches require more extra knowledge for effective transfer. Fernández and Veloso [5] proposed Policy Reuse Q-learning (PRQL) and Policy Library rebuilding through Policy Reuse (PLPR) where a learning agent is equipped with a library of previous policies to facilitate exploration, as they enable the agent to collect relevant information quickly to accelerate learning. However, PRQL converges to a sub-optimal policy in some conditions since negative transfer exists, and PLPR builds a Policy Library without a clear theoretical guarantee. Li and Zhang [10] proposed an Optimal Policy Selection TL method (OPS-TL) to select a suitable policy using a multi-armed bandit (MAB) method UCB1 [8] during the online learning. However, OPS-TL needs more performance feedback to evaluate sources to selection, which takes time to lock a known suitable policy. And it also requires a manual setting of rate to learn independently.

To address the above problems, we propose a Heuristically Adaptive Policy Reuse (HAPR) framework as our contribution. This framework consists of two parts: the primary part of HAPR for Transfer Learning (HAPR-TL) and a Policy Library rebuilding part using KL-divergence (PLKL) as secondary. HAPR-TL reuses a policy all out during learning until a state called *subgoal* is reached, where the policy is selected by evading other inappropriate source policies. A subgoal of a source policy is a special state defined by using a state-value function, which is also used to filter out bad policies or parts. The idea of subgoal is from the concept proposed by Ruby and Kibler [15]. In order to make HAPR-TL persistent and become lifelong learning, the secondary PLKL method rebuilds the library once HAPR-TL finishes. It uses KL-divergence [7] to measure the distance between two policies and determine whether to add a new policy into the policy library or not. Our experimental results on the grid-world domain show our method outperforms state-of-the-art policy reuse approaches, as HAPR adaptively reuses the useful knowledge from a policy library that is equipped with various policies without redundancy.

The main contributions of this paper can be summarized as:

(1) To reuse policy in a state-level and to avoid reuse from the irrelevant part of the source policy, we draw lessons from the subgoal method and set the subgoal as a critical state to stop policy reusing.
(2) To avoid negative transfer with reusing known unsuitable source policy, we give up the UCB1 exploration. We choose the successful ratio to the goal of tasks as the evaluation of source policies and remove the poor policy from the alternative Policy Library in each episode.
(3) To make the Policy Library more representative, we use KL-divergence to rebuild the Policy Library.

The remainder of this paper is organized as follows. Section 2 introduces the background of our approach including problem formulation in 2.1 and related works in 2.2 and 2.3. Section 3 presents our approach in two subsections: Policy Reuse in 3.1 and Policy Library rebuilding in 3.2. Section 4 gives the experimental results compared with state-of-the-art methods. Section 5 concludes our approach and draws the future work.

*Corresponding author: Jianye Hao

2 BACKGROUND

2.1 Policy Reuse Problem

RL problems are usually formulated as Markov Decision Processes (MDPs), which is defined in a 5-tuple $<S, A, T, R, \gamma>$. S is a finite set of states. A is a finite set of actions. T is a stochastic state transition function ($T = S \times A \times S \rightarrow \mathfrak{R}[0,1]$). R is a stochastic reward function ($R = S \times A \times S \rightarrow \mathfrak{R}$). And γ is a discounted factor ($\gamma \in \mathfrak{R}(0,1]$). A policy π is defined as a function that specifies an appropriate action $a = \pi(s)$ for each state s. An agent following π will get a discounted total reward $W_\pi = \sum_{h=1}^{H}(\gamma^{h-1} \cdot R(s_h, a_h))$, with reward $R(s_h, a_h)$ feedback in step h. The solution of an MDP is to find an optimal policy $\pi = \arg\max_\pi \mathbb{E}[W_\pi]$ to maximize the expected value of W_π. In practice, the mean of sampled reward $\overline{W}_\pi$ is also an evaluating indicator for the algorithm performance.

To formally describe the *policy reuse problem*, a domain $\mathfrak{D}$ is defined as a sub-tuple $<S, A, T>$ of MDPs. And a task Ω is defined as a tuple $< \mathfrak{D}, R_\Omega >$ with domain $\mathfrak{D}$ and R of MDPs. A policy library L is a set of source policies $\pi_1, \pi_2, .., \pi_n$, where policy π_i is a trained policy of task Ω_i. With different task intervals in the same domain $\mathfrak{D}$, the *policy reuse problem* is to find a way to learn the optimal policy π_Ω^* of every new task Ω by reusing source policy selected from a given policy library L.

2.2 Policy Reuse Methods

Compositional Q-learning [16] first prompted the problem of learning multiple tasks in the same domain by TL. In addition, a different Policy Reuse context for the lifelong autonomous agent was described by Rosman *et al.* [13]. PRQL [5] optimize gave a classic Policy Reuse framework to solve this problem. PRQL reuses policy, which is selected from the Policy Library with its own policy being trained following the soft-max (Boltzmann distribution) strategy, as the training policy in a certain probability. A current method OPS-TL [10] optimizes policy selection method of PRQL by using MAB method of UCB1 [8]. However, the exploration rate of OPS-TL is also increased exponentially without effective reuse in a training episode. And with no contribution, the UCB1 method sometimes selects a known poorer policy for exploration.

Reuse with exploration leads to that PRQL and OPS-TL can't quickly learn a task similar to the source task. And the reserved selection makes PRQL and OPS-TL cannot get rid of the condition of reusing unsuitable policy quickly, which leads to negative transfer at the beginning. By contrast, Our Policy Reuse method HAPR-TL will lock the selection of the most suitable policy and fully reuse the useful part of the policy.

2.3 Policy Library Rebuilding Methods

Policy Library rebuilding method provides source policies to serve future Transfer Learning and responds to the domain structure to a certain extent.

PLPR proposed by Fernández and Veloso is the only Policy Library rebuilding method that is associated with Policy Reuse. However, PLPR lacks intuitive explanation. For library rebuilding and allowing the mechanics of the environment to be learned, a parallel learning method is proposed by Ollington and Vamplew [11]. However, problems don't arise just right together in reality.

In addition, the method of Earth Mover's Distance (EMD) [14] using Wasserstein Metric can also be used to present the dissimilarity between policies as an improved method of PLPR. The recent work on representing similarity between distributions was proposed by Song *et al.*, which defines a distance between MDPs using EMD [17]. However, computing EMD needs a high time complexity.

Our Policy Library rebuilding method PLKL optimizes PLPR with the theoretical support of KL-divergence [7]. So that PLKL can further solve sequential tasks by using previous source policies as PLPR does. PLKL computes the KL-divergence in both directions of each two policies as the criteria for rebuilding the policy library simply and sufficiently.

3 APPROACH

Our approach addresses the *policy reuse problem*. It starts with a Policy Library L, and it uses a policy reuse method to learn an optimal policy of a target task with a source policy, which is selected by a policy selection method from L. Our approach has also rebuilt L for the next task to learn. In this section, we give our policy reuse framework HAPR to solve the Policy Reuse problem, which concludes its main part HAPR-TL for policy reuse, with a library rebuilding method PLKL to assist it, as shown in Algorithm 1.

Algorithm 1 HAPR

```
Require: Policy Library $L$
loop
    get a new task $\Omega$
    $\pi_\Omega \leftarrow$ HAPR-TL($\Omega, L$)
    $L \leftarrow$ PLKL($L, \pi_\Omega$)
end loop
```

In Algorithm 1, given a policy library L and a new task Ω, HAPR-TL learns an optimal policy of Ω by fully reusing policies selected from L. And PLKL rebuilds the Policy Library L once obtaining the learned policy. The details of each part are shown in the following section 3.1, and section 3.2, which are two main components of our transfer learning framework.

3.1 Policy Reuse with Selection for TL

Considering that the agent cannot foresee whether the knowledge of source policy is suitable for the target task to transfer, our approach focuses on the way to select the useful part from given source policies for reuse. In this section, we introduce our HAPR-TL method in two parts following: Policy Selection and Policy Reuse.

Policy Selection in HAPR-TL. An intention of Policy Reuse is to reuse useful parts of source policies not to well-train before. In our approach, the source policy selection method has three purposes.

(1) To quickly lock the most suitable policy for the target task.
(2) To get the useful part of a selected source policy to reuse.
(3) To stop reuse once the target policy performs as good as the source policy.

In our algorithms: S stands for the state set containing all the available states in our domain; L stands for the Policy Library; and L' stands for the policy set for selecting source policies. And $|S|$, $|L|$

Algorithm 2 HAPR-TL(Ω, L)

```
Require: States Set $S$; target task $\Omega$ with its goal state $s_G$; source
    policies in Policy Library $L$ with the $Q$-function, the $C$-functions
    and the subgoal set for each policy
 1: Initialize Target Policy $\pi_\Omega$: $\forall Q(s,a) \leftarrow 0, \forall C(s) \leftarrow 0$
        $\tau \leftarrow [\,]$; $L' \leftarrow L$; $\acute{s} \leftarrow s_G$; $n_i \leftarrow 1, p_i \leftarrow 0.5 \quad \forall i = 1..|L|$
 2: for $n = 1..N$ do
 3:     if $s_G \notin \tau$ or $\forall \acute{s} \notin \tau$ for subgoals $\acute{s}$ of any policy in $L'$ then
 4:         if $n > 1$ then
 5:             $p_k \leftarrow \frac{p_k \cdot n_k}{n_k+1}$
 6:         end if
 7:         $k \leftarrow$ the next index $i$ of $\pi_i \in L'$
 8:     else
 9:         $p_k \leftarrow \frac{p_k \cdot n_k + 1}{n_k+1}$
10:         for $\forall \pi_i \in L'$ do
11:             if $2 \cdot p_i < \max_{\pi_j \in L'}(p_j)$ then
12:                 $L' \leftarrow L' - \{\pi_i\}$
13:             else
14:                 $g(\pi_i) \leftarrow \sum_{s \in \tau[\acute{s}, s_G]}(\min\{0, C_i(s) - \varsigma\})$
15:             end if
16:         end for
17:         $k \leftarrow \arg_i \max\{g(\pi_i) | \pi_i \in L'\}$
18:     end if
19:     $n_k \leftarrow n_k + 1$
20:     Get uniform random Initial State $s_0 \in S$
21:     Get the subgoal $\acute{s}$ of $\pi_k$ by $\tau$ and the performance of $\pi_\Omega$.
22:     $[\pi_\Omega, \tau] \leftarrow \pi\_\text{reuse}(\pi_k, \pi_\Omega, s_0, \acute{s}, s_G, \tau)$
23: end for
24: return $\pi_\Omega$
```

and $|L'|$ are the numbers of elements in those sets. ς is a mean-value threshold of the C-function ($\varsigma := \frac{1}{|S|}$).

Algorithm 2 takes N episodes to train the policy for Ω. In each episode, our selection method first selects a source policy to reuse (Line: 3-18) and then revise the state of subgoal (Line: 21). After that is our Policy Reuse method (Line: 22; shown in Algorithm 3).

For each episode, τ is the sampled state trajectory, which records all the states passed in the previous episode in order. Only if τ has reached the goal of the target task and also reaches at least one subgoal of any source policy in L' (Line: 8), our *policy selection* mechanism will really be implemented (Line: 10-17). Otherwise, it will select a policy and a subgoal in turn (Line: 3,7).

A success rate p to the goal of a source policy is adjusted in every episode to indicate the reliability of the policy (Line: 5, 9). Our selection method stops reusing of poor policy (with a lower p) by removing the policy from the library L' for selecting (Line 11-12).

For each source policy π_{src}, we first require a *subgoal set* to select a subgoal $\acute{s}$ for our Policy Reuse method (Line: 21, 22). The subgoal set includes the subgoals and the goal of π_{src}. If some states in the subgoal set exist in the trajectory τ, our method will revise the current subgoal $\acute{s}$ to one of those states last appeared in τ. Otherwise, it will select a subgoal of π_{src} in turn. Furthermore, if the target policy π_Ω has a higher mean value of total reward $\overline{W}_\pi$ than any source policy π_{src} in a testing batch, it will degenerate $\acute{s}$ to the initial state s_0 of the current episode.

In our approach, we propose a C-function to present the importance of states. If a source policy only contains Q-function, we will add C-function to the policy. The values of C can be figured out by generating a series of trajectories guided by Q. C updates every step in a sampled trajectory. In a step from state s' to s:

$$C(s) \leftarrow (1-\alpha) \cdot (C(s)+1) + \alpha \cdot C(s') \tag{1}$$

where C is updated in the form of the linear combination of $(C(s)+1)$ and $c(S')$. Stepsize α shares the same value with the stepsize of Q iteration in our approach. Finally, C needs to be normalized. We use C-function for both comparing the source policies and obtaining the subgoal set.

For the trajectory reaching the goal and the subgoal in the previous episode, we calculate the cumulative value of C over the threshold ς for every source policy in L' to compare and select an outstanding source policy π_k (Line: 14, 17):

$$k \leftarrow \arg_i \max \sum_{s \in \tau[\acute{s}, s_G]} (\min\{0, C_i(s) - \varsigma\}) \tag{2}$$

where $\tau[\acute{s}, s_G]$ is the set including all the state from $\acute{s}$ to s_G that the trajectory τ passes through.

Sharing the same idea with a *subgoal* method [6], we use C-function of each source policy to figure out its subgoal set as Algorithm2 requires. In the h^{th} step of a sampled trajectory based on a source policy, that the state s_h is a *subgoal* of the policy subjects to:

$$\Delta C(s_h) > \rho \cdot \max(\delta_C, \Delta C(s_h + 1)) \tag{3}$$

where

$$\Delta C(s_h) = C(s_h) - C(s_{h-1}) \tag{4}$$

In Formula 3 and 4, δ_C and ρ are two positive thresholds, which indicates that $\Delta C(s_h) > 0$ for the subgoal s_h. For each source policy, our approach only adopts its subgoals in a small number, which number can be effectively controlled by adjusting these two thresholds.

The C-function can also be used to examine how frequently the training has worked in a state, and also to figure out the KL-divergence between policies.

The HAPR-TL method finishes with returning the policy π_Ω derived by Q with C included. Our Selection method uses the notion of subgoal and C-function to avoid negative transfer from source policies. This method is constructed to match the following Policy Reuse method.

Policy Reuse of HAPR-TL. As we hand over the duty of sifting the useful part of source policy to the Policy Selection method, in Algorithm 2, we character our Policy Reuse method to completely reuse the given policy. The subgoal $\acute{s}$ selected in Algorithm 2 is used as a signal in our Policy Reuse method shown in Algorithm 3. Our method keeps reusing the source policy until a given subgoal is reached and then turns to learn with its own policy.

To reuse policy fully in H steps, we keep reusing policy until a provided subgoal $\acute{s}$ reached (Line: 3-4). After the step arriving $\acute{s}$ in an episode: if we have arrived the goal s_G of the target task in the previous episode, we will use the ϵ-greedy method to correct our target policy; otherwise we use the random policy to enhance our exploration (Line: 5-10).

A new state trajectory τ_{neo} consists of the initial state s_0 and all the state arrived after an action in each step (Line: 1,12). When

```
Algorithm 3 π_reuse ($\pi_{src}, \pi_\Omega, s_0, \acute{s}, s_G, \tau$)
Require: Source Policy $\pi_{src}$; Target Policy $\pi_\Omega$ with its $Q$-value
    function and $C$-value function; Initial State $s_0 \in S$; State of
    subgoal $\acute{s}$; Goal State $s_G$; last State Trajectory $\tau$
 1: Initialize $\tau_{neo} \leftarrow [s_0]$
 2: for $h = 1..H$ do
 3:   if $\acute{s} \notin \tau_{neo}$ then
 4:     $a_{h-1} \leftarrow \pi_{src}(s_{h-1})$
 5:   else
 6:     if $s_G \in \tau$ then
 7:       $a_{h-1} \leftarrow \epsilon\_greedy(\pi_\Omega(s_{h-1}))$
 8:     else
 9:       Get uniform random $a_{h-1} \in A(s_{h-1})$
10:     end if
11:   end if
12:   Get state $s_h$ after the action $a_{h-1}$ and put $s_h$ in $\tau_{neo}$.
13:   Update $Q$:
      $Q(s_{h-1}, a_{h-1}) \leftarrow$
        $(1-\alpha) \cdot Q(s_{h-1}, a_{h-1}) + \alpha \cdot (r_h + \gamma \cdot max_a(Q(s_h, a)))$
14:   if $s_{h-1} \notin \tau_{neo}$ then
15:     $C'(s_h) \leftarrow (1-\alpha) \cdot C(s_h) + \alpha \cdot C(s_{h-1}) + 1$
16:   end if
17:   if $s_h = s_G$ then
18:     Update $C$: $C \leftarrow C'$
19:     exit
20:   end if
21: end for
22: Normalize $C$ with sum of 1
23: return $\pi_\Omega, \tau_{neo}$
```

getting a new state s after an action from s', Q-function is updated. The iteration of C-function works at the same time with Q, but it is just temporarily updated:

$$C'(s) \leftarrow (1-\alpha) \cdot C(s) + \alpha \cdot C(s') + 1 \tag{5}$$

Formula 5 is slightly different from Formula 1 because $C(s')$ is not really updated. Considering to update C more unbiased, $C(s)$ is only temporarily updated in the first time reaching a state s during the current episode. And C is really updated only if the current episode finally arrives to the goal state S_G (Line: 14-18). Algorithm 3 finished if s_G is arrived or time step H is up. And C-function also needs to be normalized here (Line: 22).

As our Policy Reuse method maximizes the reuse of the policy part selected, the method will have a good performance at the very beginning if the part reused is useful, or it will feedback a bad performance in time and the algorithm will avoid selecting the policy having a negative correlation with the target task. And it also can prevent the unbalanced sampling of trajectories in training, so that the new C generated can be used as a C-function within the source policy for the PLKL method in the next subsection.

3.2 Policy Library rebuilt with KL-divergence

Follow the previous work settings [5], the *policy reuse problem* often considers the situation that an agent is equipped with a policy library, although HAPR-TL can also learn with itself. In addition, a policy library can indirectly represent the dynamics of the environment, which also helps the learning process of the agent.

In this section, we give a policy library rebuilding method to ensure the independence among source policies. First, to ensure that each policy in the Policy Library is unique and independent with each other, we use KL-divergence (relative entropy) [7] as a simple criterion to measure the dissimilarity between two policies. Our PLKL method takes the advantages of the KL-divergence of the C-functions of a source policy and the target one in both directions shown in Algorithm 4.

```
Algorithm 4 PLKL($L, \pi_\Omega$)
Require: Policy Library $L$ with $C$-value functions; Target Policy
    $\pi_\Omega$ with its $C$-value function
 1: for $\pi_i \in L$ do
 2:   $D_{KL} \leftarrow 0$; $D_{KL\text{–}inv} \leftarrow 0$
 3:   for $\forall s \in S$ do
 4:     if $C_\Omega(s) > 0$ and $C_i(s) > 0$ then
 5:       $D_{KL} \leftarrow D_{KL} + C_i(s) \cdot log(\frac{C_i(s)}{C_\Omega(s)})$
 6:       $D_{KL\text{–}inv} \leftarrow D_{KL\text{–}inv} + C_\Omega(s) \cdot log(\frac{C_\Omega(s)}{C_i(s)})$
 7:     end if
 8:   end for
 9:   if $D_{KL} < \delta$ then
10:     return $L$
11:   else if $D_{KL\text{–}inv} < \delta$ then
12:     $L \leftarrow L - \{\pi_i\}$
13:   end if
14: end for
15: return $(L \cup \{\pi_\Omega\})$
```

We calculate the KL-divergence between the source policy and the target policy to measure the former can be replaced by the latter, shown in Formula 6:

$$D_{KL} = \sum_{\forall s \in S, \forall C(s) > 0} (C_{src}(s) \cdot log(\frac{C_{src}(s)}{C_{tar}(s)})) \tag{6}$$

where the source policy's C-function C_{src} and the target policy's C-function C_{tar} can be considered as normalized distributions of the importance of states in their respective tasks. A large value of D_{KL} in Formula 6 indicates that the original policy is not similar with the target policy. In Algorithm 4, if every D_{KL} exceeds a threshold $\delta_D = \delta$ ($\delta \in \Re(0,1]$), the new policy will be joined in the Policy Library L (Line 15). We refer the PLKL only considering the uni-direction KL-divergence (without Line: 11,12) as the uni-direction PLKL (uni-KL) method.

Similarly, the inverse KL-divergence can find out whether the target policy can replace a source policy:

$$D_{KL\text{–}inv} = \sum_{\forall s \in S, \forall C(s) > 0} (C_{tar}(s) \cdot log(\frac{C_{tar}(s)}{C_{src}(s)})) \tag{7}$$

A small value of $D_{KL\text{–}inv}$ value in Formula 7 indicates that the target policy can replace the source policy. In Algorithm 4, besides joining the target policy in the Policy Library L, when the value of $D_{KL\text{–}inv}$ is lower than δ_D while D_{KL} is not, the new policy is decided to replace the source policy in L (Line: 11,12,15). We refer the whole

PLKL method as the bi-direction PLKL (bi-KL) method. According to the asymmetry of KL-divergence, this method can remove similar policies from the source Policy Library.

Our PLKL method ensures that each policy is independence from each other with a theoretical guarantee, as KL-divergence can describe the difference from one distribution to another. KL-divergence is not the only option for the requirement, Bhattacharyya distance [1] or JS-divergence [4] can also be used here.

4 EXPERIMENTS

In this section, we provide experimental results on a grid-world navigation domain. We first give some heat-maps to show the feasibility of the C-function in our results. Then we verify that our Policy Reuse method HAPR can transfer quickly and avoid negative transfer, compared with the related methods including ϵ-greedy Q-learning, PRQL [5] and OPS-TL [10]. We also compare our PLKL with libraries rebuilt by PLPR [5].

4.1 Experimental Setting

Our experiments use a 24×21 grid-world navigation domain with 50 sequential tasks, which are represented by their *goals* in Figure 1.

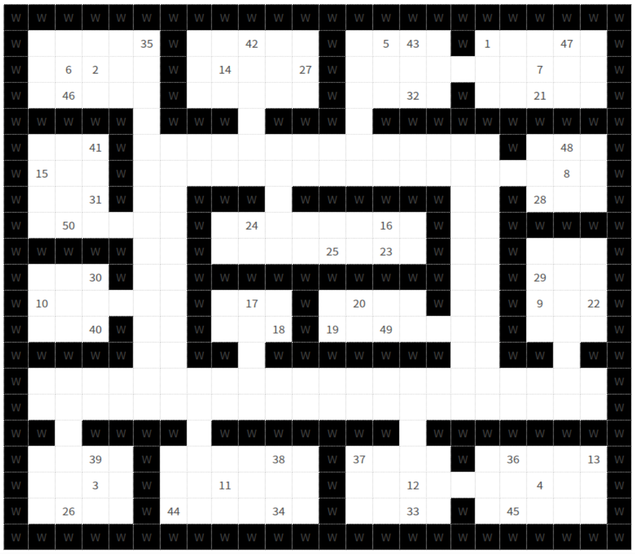


Figure 1: The domain of Grid-world2006 with 50 tasks

In Figure 1, the type of grid can only be *normal*, *wall* or *terminal*, whose functions are introduced below.

Normal grid is the only grid in which the agent can form a legal state in the MDP process. In our experiments, an agent will be randomly generated in a *normal* grid to start a navigation episode with a number of steps. In each step, the agent will choose one direction from *east*, *north*, *west* and *south* with a distance to move as an action. After an action, the agent will access to a new grid. If the new grid is the *wall*, the agent will be transferred back to the last position and waste one step. If the new grid is the *terminal*, a reward of arriving the *goal* state (only arriving *terminal* has reward) will be received and the current episode should be terminated. The *terminal* must be fixed in our *goal*-oriented task.

For ease of description, we named every grid by its column and row start from "Grid(0,0)" on the top-left, such as "Grid(3,2)" refers to the *terminal* grid of task 2 in Figure 1. Without loss of generality, we set the length of grids' edge to 1. And we represent the position of agent within a two-dimensional continuous coordinates (x_a, y_a), where $x_a \in \Re[0, 24)$ and $y \in \Re[0, 21)$. The agent is in grid(x,y) when $\lfloor (x_a, y_a) \rfloor = (x, y)$, where $x \in \mathfrak{N}[0, 24)$ and $y \in \mathfrak{N}[0, 21)$. Each action can change one coordinate of the agent's position in length 1 in a direction. To *east* increase the "x_a"; to *north* decrease the "y_a"; to *west* decrease the "x_a" and to *south* increase the "y_a". The actual arrival position is affected by an error following a uniform distribution in a range of $(-0.20, +0.20)$.

This environment has $|S| = 301$ accessible *normal* states. We set $\alpha = 0.05$, $\gamma = 0.95$ for Q-function & C-function update for all method in experiment. And $\epsilon = 0.90$ as the exploit rate of the ϵ-greedy method. The parameters of the comparison methods including PRQL[5] and OPS-TL[10] are consistent with the best in those methods. Each method will be trained in $N = 4000$ episodes and at most $H = 100$ steps within an episode. And for each $N' = 100$ episodes, there is a set of test episodes to evaluate the performance of the target policy trained.

4.2 Experiment Results

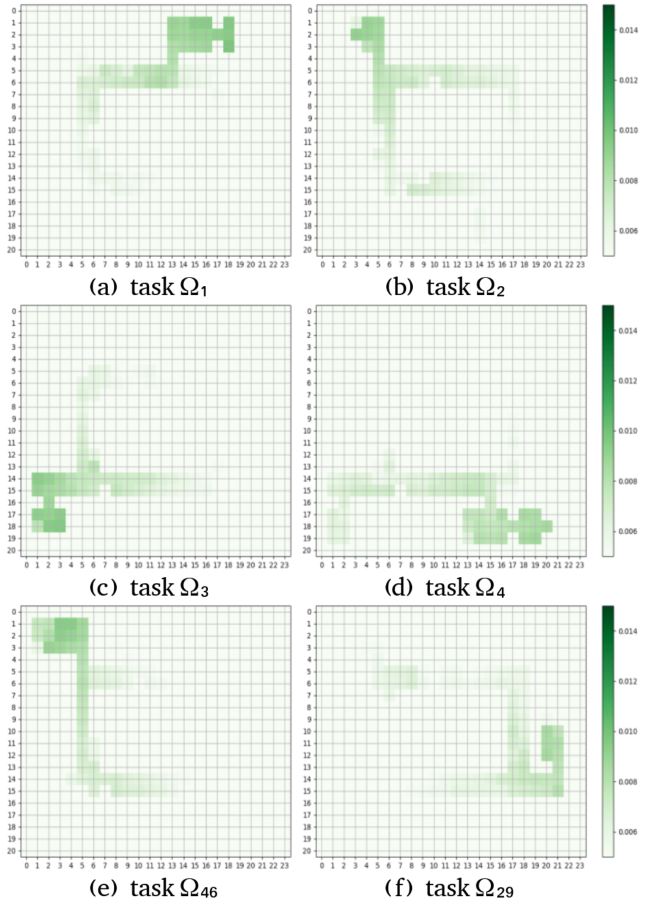
In our experiments, the well-trained policies of tasks Ω_1, Ω_2, Ω_3 and Ω_4, whose *goals* are shown in Figure 1, are chosen into an initial source Policy Library $L_{init} = \{\pi_1, \pi_2, \pi_3, \pi_4\}$.

Feasibility of C-function. In the first experiment, we choose tasks Ω_{46} and Ω_{29} shown in Figure 1 as target tasks. We compare the C-function of them with the tasks in L_{init}.

It is intuitive to see that the tasks corresponding to the same room (Ω_{46} and Ω_2) in Figure 1 are similar. And the heat-maps in Figure 2 shows that such similar tasks have great similarity with their C-functions, which is normalized by the sum of 1 in this experiment. Figure 1 also shows that Ω_{29} didn't have any similar task from L_{init}. This result manifests that the C-function can be used as a representative feature of a task. According to this result, we respectively choose Ω_{46} and Ω_{29} as the target tasks in the second experiment and the third experiment.

In addition, we get the subgoals of each source policy in preparation for the next experiments. Instead of setting values for δ_C and ρ in Formula 3, we sort the states directly according to the form of Formula 3 and choose the first b ($b = 2$) states as subgoals for each source policy π_{src}. First, we sample several trajectories according to π_{src} and figure out $\Delta C(s)$ and $\Delta C(s')$ for each state s and the next s' in every trajectory. Then, we find out every subgoal s with $\Delta C(s) > 0$ and $\Delta C(s') \leq 0$. We rank them according to their $\Delta C(s)$. If their number is more than b, we take the first b states of them as subgoals in π_{src}'s subgoal set. Otherwise, until the number of subgoals reaches b, we will keep picking up the state with the highest value of $\frac{\Delta C(s)}{\Delta C(s')}$ from the rest with $\Delta C(s) > 0$.

In our experiment, the subgoal sets for the source policies in Policy Library L_{init} are shown in Table 1. For each task, the states in subgoal set consists of two subgoals and the goal of the task.



(a) task Ω_1 (b) task Ω_2 (c) task Ω_3 (d) task Ω_4 (e) task Ω_{46} (f) task Ω_{29}

Figure 2: C-functions of $\Omega_1, \Omega_2, \Omega_3, \Omega_4, \Omega_{46}$ and Ω_{29}

Table 1: Subgoal and goal for each task in L_{init}

Task	subgoal $\acute{s}_1$	subgoal $\acute{s}_2$	goal s_G
Ω_1	Grid(13,5)	Grid(16,2)	Grid(18,1)
Ω_2	Grid(6,6)	Grid(5,5)	Grid(3,2)
Ω_3	Grid(5,15)	Grid(2,15)	Grid(3,18)
Ω_4	Grid(17,15)	Grid(18,18)	Grid(20,18)

HAPR-TL with a similar task. In the second experiment, we choose the task Ω_{46} shown in Figure 1 as the target task. It obviously has a similar task Ω_2 in the Policy Library L.

Figure 3 shows the learning curves of HAPR-TL in our approach, OPS-TL, PRQL and ϵ-greedy QL top-down when solving task Ω_{46}. The learning curve is generated by the average discounted reward $\overline{W}$ of each method's on-policy testing, which executed 10 times after every 100 episodes from 100 to 4000. Error bars of standard deviations is shrinking to half.

In Figure 3, the average reward W of HAPR-TL is greater than 0.3 at starting with the first 100 episodes converges quickly in about 1000 episodes to a value more than 0.4. Compared with PRQL and OPS-TL, HAPR has the advantage at "Jumpstart", "Asymptotic Performance", "Time to Threshold" and other evaluate metrics proposed by Taylor and Stone [20]. Our method learns quickly mainly because we fully reuse the source policy with a quick selection. However, the other method have slow selection, and their reuse rate updates as $\varphi \longleftarrow \gamma \cdot \varphi$ with $\gamma = 0.95$ in every step, which leads to a low learning rate φ even in the tenth step ($\varphi(10) = 0.95^{10} < 0.60$).

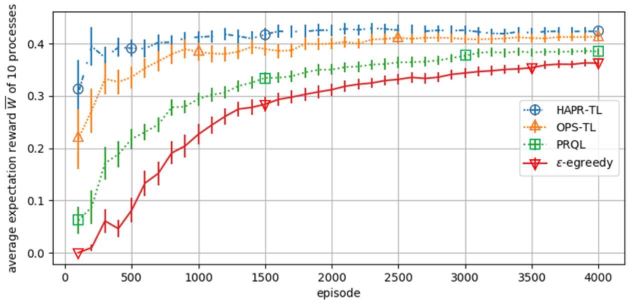


Figure 3: Average discounted rewards with suitable π_{src}

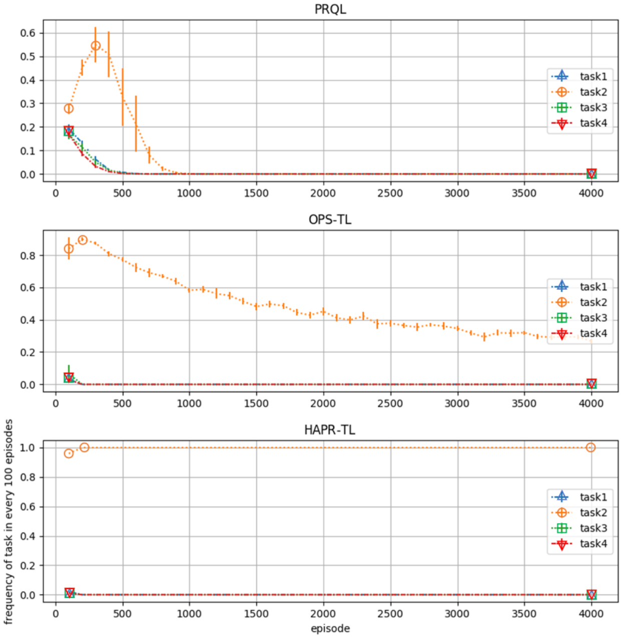


Figure 4: Frequency of reuse from each source task to Ω_{46}

Figure 4 shows frequency curves of the reuse rate of each source policy in the Policy Library L, where the curve of reusing policy of Ω_2 is obviously higher than other curves in all three algorithms as HAPR-TL, OPS-TL, and PRQL. Error bars represent standard deviations.

In Figure 4, HAPR-TL quickly locked the source policy at the beginning with the highest rate near 100%. It shows that our selection method is effective to select the right policy. In HAPR-TL, the exploitation rate for reusing policy of Ω_2 did not go down as in

PRQL or in OPS-TL, because our method has defined the subgoal for *policy reuse problem* and actually learns independently when the subgoal degenerate to the initial state in an episode.

HAPR-TL without any similar task. In the third experiment, we choose task Ω_{29} as the target task, since it has no similar task in the Policy Library mentioned in the first experiment.

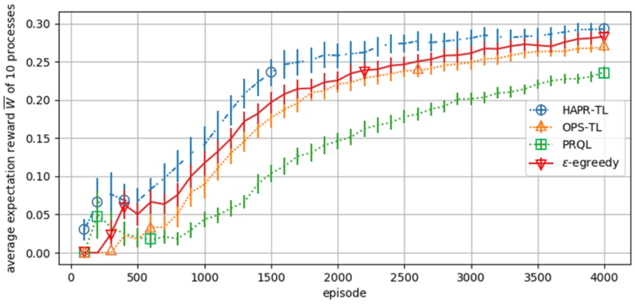


Figure 5: Average discounted rewards with unsuitable π_{src}

Figure 5 shows the learning curve of HAPR-TL, ϵ-greedy QL, OPS-TL, and PRQL of Ω_{29} in top-down order. The curve in Figure 5 is generated in the same way as in Figure 3.

In Figure 5, HAPR-TL can be found to basically exceed ϵ-greedy QL, while OPS-TL is just close to ϵ-greedy QL. PRQL performs the worst. Our method performs well at the beginning mainly because it gets rid of the bad part of policies to reuse since all source policies are not able to completely reuse.

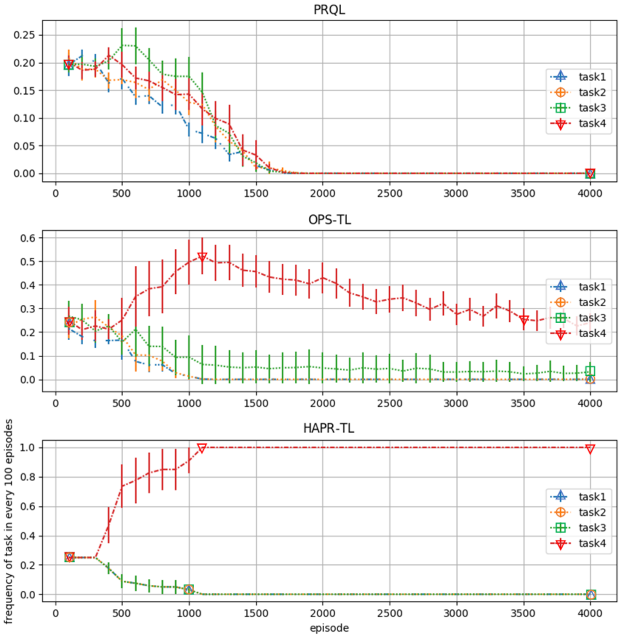


Figure 6: Frequency of reuse from each source task to Ω_{29}

Figure 6 shows frequency curves of each source policy for learning Ω_{29}, in the same way as Figure 4. The curve of reusing policy of Ω_4 is obviously higher than other curves in HAPR-TL and OPS-TL.

In Figure 6, without a suitable source policy, reusing whole source policy quickly abandoned as the frequency curves of reusing each policy bifurcates in about 300 episodes. However, OPS-TL needs 500 episodes to start this. Our method is better than OPS-TL and PRQL in the absence of the source policy with its stability since we still reuse part of policy π_4.

Therefore, experiments 2 and 3 empirically demonstrate that HAPR-TL significantly lowers the sample complexity of reaching convergence.

Policy Library rebuilt with KL-divergence. In the last experiment, we first show the Policy Library rebuilt using KL-divergence in uni-direction and bi-direction of PLKL by executing 50 different tasks in Figure 1 sequentially performed with Library L_{init} at starting.

To show that using the KL-divergence as a similarity measure between policies performs better than the way used in PLPR, we review the results of the Rebuilt Library in PLPR firstly. After executing each task, it will append the new policy to the Policy Library $if\ and\ only\ if\ max_i(W_\Omega(\pi_i)) < \delta \cdot W_{task}(\pi_\Omega)$. Figure 7 shows its result with its known best parameter $\delta = 0.25$ [5].

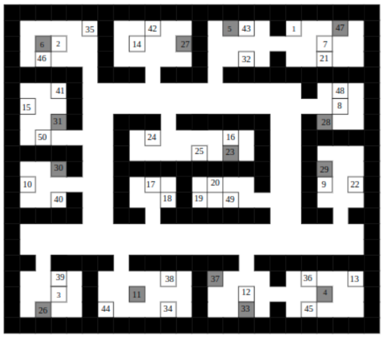


Figure 7: Shadow tasks in Library Rebuilt via PLPR

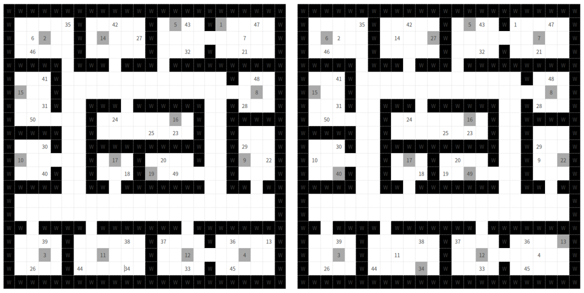
In Figure 7, the source task distribution generated by PLPR is not uniform: two source policies exist in one room; some rooms have no source policy. The two bad conditions cannot be alleviated together by only adjusting the parameter δ.

Then we show our results of the Policy Library rebuilt by using PLKL. The PLKL method is good for screening the unique policy of each room in both uni-direction way and bi-direction way, as shown in Figure 8,

In Figure 8, to compare the Policy Library generated by the bi-direction KL-divergence (bi-KL) with the uni-KL: some source policies have been replaced by new policies in a way. If the threshold is higher, the source Policy Library after the reconfiguration will be incomplete in some conditions, and if the threshold is lower, the source policy will be replaced many times, of course, this does not affect the availability of the Policy Library.

In this experiment, we also show the comparison of learning effect among different Policy Libraries used for solving a new task. We choose task Ω_{47} to learn. And we set different conditions of initial policy library:

(1) a normal Policy Library L_{init} (also the Library with uni-KL)



(a) uni-direction PLKL (b) bi-direction PLKL

Figure 8: Shadow tasks in Library Rebuilt via HAPR-TL

(2) a fragmentary Policy Library $L' = \{\pi_2, \pi_3, \pi_4\}$
(3) a policy library $L^* = \{\pi_2, \pi_3, \pi_4, \pi_7\}$ generated by bi-KL

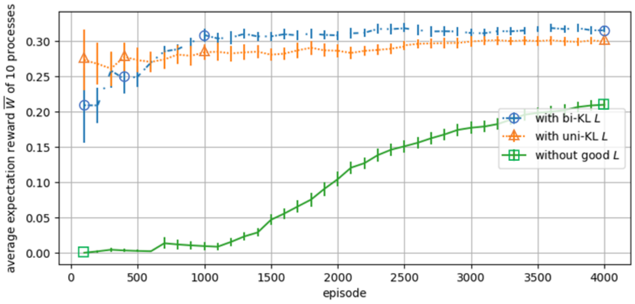


Figure 9: Expected W with uni-KL and bi-KL and None

Figure 9 shows that reuse from a Policy Library with abundant policies performs better than not, which affirms the necessity of our PLKL method. In addition, it shows that PLKL in a bi-direction way can optimize its effect more. Both the methods of PLKL are stable, as they solve the problem well.

5 CONCLUSION AND FUTURE DIRECTIONS

This work focuses on multi-task transfer in RL. We propose a new Policy Reuse framework HAPR, including the method HAPR-TL for policy reuse and the method PLKL for policy library rebuilding. Our HAPR-TL method enhances reuse efficiency and avoids negative transfer. HAPR-TL optimizes the Policy Selection method by evading known unsuitable policies and the unsuitable part of source policy by giving some subgoals and then fully reuses the source policy selected until a given subgoal of that source policy is arrived. In contrast to previous work, our work reuses the policy quickly at the state-level and avoids negative transfer. In addition, we have the PLKL method to provide a trenchant Policy Library for the next learning tasks, with more theoretical basis than previous algorithms. Compared with the relevant methods, our methods have the top performance in all the experiments designed in the navigation domain.

For future work, a major improvement direction is to automatically generate parameters. Another direction is that how the selection method can decide which part of policy to reuse for more efficient policy transfer. Furthermore, we will extend our framework to deep RL domain. In the directions of the above improvement, we will also tests our method in other domains and applies in practical problems.

REFERENCES

[1] A. Bhattacharyya. 1946. On a Measure of Divergence between Two Multinomial Populations. *SankhyÄĄ: The Indian Journal of Statistics (1933-1960)* 7, 4 (1946), 401–406.

[2] Tim Brys, Anna Harutyunyan, Matthew E Taylor, and Ann Nowé. 2015. Policy transfer using reward shaping. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 181–188.

[3] Bruno N Da Silva and Alan Mackworth. 2010. Using spatial hints to improve policy reuse in a reinforcement learning agent. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 317–324.

[4] Ido Dagan, Lillian Lee, and Fernando Pereira. 1997. Similarity-Based Methods For Word Sense Disambiguation. *Proceedings of the Association for Computational Linguistics* 6493, 10 (1997), 56–63.

[5] F. Fernández and M. Veloso. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, Vol. 58. 720–727.

[6] Sandeep Goel and Manfred Huber. 2003. Subgoal Discovery for Hierarchical Reinforcement Learning Using Learned Policies.. In *Sixteenth International Florida Artificial Intelligence Research Society Conference, May 12-14, 2003, St. Augustine, Florida, Usa*. 346–350.

[7] S. Kullback and R. A. Leibler. 1951. On Information and Sufficiency. *Annals of Mathematical Statistics* 22, 1 (1951), 79–86.

[8] T. L Lai and Herbert Robbins. 1985. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics* 6, 1 (1985), 4–22.

[9] Romain Laroche and Merwan Barlier. 2017. Transfer Reinforcement Learning with Shared Dynamics. In *AAAI Conference on Artificial Intelligence*. 2147–2153.

[10] Siyuan Li and Chongjie Zhang. 2018. An Optimal Online Method of Selecting Source Policies for Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16800/16556

[11] Robert Ollington and Peter Vamplew. 2005. Concurrent Q-learning: Reinforcement learning for dynamic goals and environments. *International Journal of Intelligent Systems* 20, 10 (2005), 1037–1052. https://doi.org/10.1002/int.20105

[12] S. J. Pan and Q. Yang. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[13] Benjamin Rosman, Majd Hawasly, and Subramanian Ramamoorthy. 2016. Bayesian policy reuse. *Machine Learning* 104, 1 (2016), 99–127.

[14] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision* 40, 2 (2000), 99–121. https://doi.org/10.1023/A:1026543900054

[15] David Ruby and Dennis F. Kibler. 1989. Learning Subgoal Sequences for Planning. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*. 609–614. http://ijcai.org/Proceedings/89-1/Papers/097.pdf

[16] Satinder Pal Singh. 1992. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning* 8, 3-4 (1992), 323–339.

[17] Jinhua Song, Yang Gao, Hao Wang, and Bo An. 2016. Measuring the Distance Between Finite Markov Decision Processes. In *International Conference on Autonomous Agents and Multiagent Systems*. 468–476.

[18] Halit Bener Suay, Tim Brys, Matthew E Taylor, and Sonia Chernova. 2016. Learning from demonstration for shaping through inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 429–437.

[19] R. S. Sutton and A. G. Barto. 1998. *Reinforcement Learning: An Introduction*. Cambridge: MIT Press.

[20] M. E. Taylor and P. Stone. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research* 10, 10 (2009), 1633–1685.