Policy Transfer in Reinforcement Learning: A Selective Exploration Approach

Akshay Narayan
School of Computing
National University of Singapore, Singapore
anarayan@comp.nus.edu.sg

Tze Yun Leong
School of Computing
National University of Singapore, Singapore
leongty@comp.nus.edu.sg

ABSTRACT

We propose a new method for policy transfer in reinforcement learning problems that require fast responses adapted from incomplete, prior knowledge. We consider the setting where the source and target tasks share similar objectives but differ in the transition dynamics, e.g., for a robotic agent operating in similar but challenging environments, such as care homes and hospital wards. Policy reuse is effected by identifying the sub-spaces that are different in the target environment, where the source knowledge is insufficient. We present an exploration strategy that selectively and efficiently explores the target task. We demonstrate the flexibility of the proposed method by incorporating different exploration mechanisms for learning. We empirically show that our method performs better in terms of jump starts and average rewards, as compared to the state-of-the-art policy reuse methods.

KEYWORDS

Reinforcement learning; Policy transfer; Transfer in RL

1 INTRODUCTION

Using past knowledge to bootstrap learning reduces the number of samples required to learn an optimal policy in reinforcement learning (RL). Knowledge reuse to quickly adapt to new environments is commonly effected through transferring policies, state-action trajectories, action-value functions, and other components in the framework [28]. Transfer mechanisms can improve various aspects of learning including minimizing exploration in the target task, providing a better jump start to the learning, and reducing the learning time [35].

We propose a selective exploration and policy transfer algorithm (SEAPoT) to solve the class of problems that would benefit from reusing previous knowledge maximally, while adapting to changes where the prior knowledge is insufficient. Many real-life applications like assistive robots in geriatric care homes and hospital wards would benefit from our method. We focus on the settings where the source and target tasks differ in the transition dynamics and/or reward functions. We define similarity using the distance between the corresponding state-action transition distributions of the two tasks. The similarity in the environments is captured in a shared state-action space, and the difference is represented in a distribution of environmental elements leading to different transition dynamics.

The policy transfer approach results in strictly less information being transferred across the tasks. The agent follows the source policy until a change is detected in the environment. Limited exploration is performed in the target task to circumvent the surrounding region of the changed point until a *known state* with respect to the source task is reached. The agent then continues to follow the source task policy from this known state. We propose a new approach to identify changes across tasks and construct sub-space to focus on the most relevant partition of the task's state space to limit the exploration.

Current policy reuse methods include gathering expert suggestions related to spatial hints [10], probabilistic exploration [16], extracting partial policies using the structure of the policy space [22] and reward shaping to define target task policy from source policy [7]. Unlike previous methods, our approach does not require expert suggestions; exploration is kept minimal in the target task, and no knowledge of the source policy structure is required for effective transfer. Another class of policy reuse mechanism follows a reactive approach, by first evaluating a policy selected from a set of learned policies, determining the returns, and choosing the best policy [1]. In contrast, our approach adapts to the changes in the task when the need arises. We adopt a change detection mechanism that obviates the explicit specification of the source and the target task.

2 BACKGROUND

A Markov Decision Process (MDP) is often used to model the underlying environment in RL. A MDP is a tuple $\langle S, A, T, R\rangle$ where S is a finite set of states, A is a finite set of actions. The transition function T describes the probability of reaching state s' when executing action a in state s, i.e., $T(s, a, s') = P(s'|s, a)$. The reward function, R, represents the one-time reward received by executing action a in state s. A policy is the mapping from states to actions ($\pi : S \longrightarrow A$); the policy which maximizes the cumulative reward (optimal policy), π^*, is the solution to the MDP.

In model-based RL, the agent first estimates the transition and reward functions, and then computes the optimal policy using these estimates. The knowledge encoded in the model can be transferred across the tasks. In this work, we adopt the popular R-MAX exploration strategy [6], where the agent constructs a fictitious model that gives maximal reward to state-action pairs that are not explored sufficiently. The R-MAX agent's optimal behavior in the fictitious model leads to a balance of exploration and exploitation in the actual model. We represent each state by a vector of features as, also known as Factored MDP [21].

Transfer learning in RL involves a set of tasks where the agent has learned the policy, the *source tasks*, and a set of tasks that are new, the *target tasks*. A task, in our case, can be informally described as an environment (modeled using MDP) with known set of goal states where the agent acts. The agent can reuse various forms of knowledge such as value functions, policies, etc., learned in the

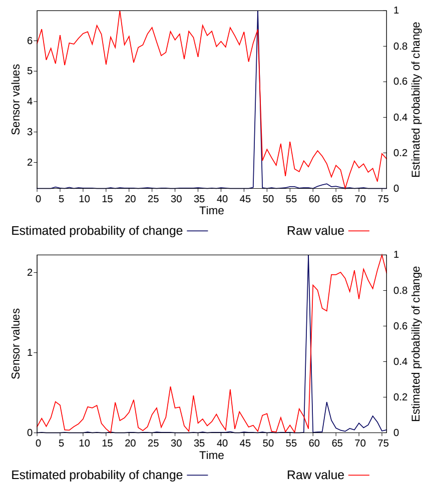


Figure 1: Change detection when an obstacle is added (top) and removed (bottom)

source in a target task [28, 35]. We focus on policy transfer in this work.

3 SELECTIVE EXPLORATION FOR POLICY TRANSFER

The agent is equipped with the solution(s) of previously solved task(s). When the agent faces a new task, it reuses the source knowledge, so long as it is applicable and reusable in the target task. The agent uses a change-detection mechanism to identify deviations from its prior knowledge. When the environment changes, the source knowledge becomes insufficient to solve the new task and the agent has to adapt to the changes detected. The agent does so by constructing a sub-space that encompasses the changed region of the target task. Learning takes place in this sub-space to obtain a policy local to the changed region. The main components in our policy transfer method (called SEAPoT) are Bayesian change detection and selective exploration.

3.1 Change detection

Changepoints split the time series data into disjoint segments such that, given a changepoint, data prior to it and data appearing after are independent of each other. Changepoints in the agent's observations in a particular state, s, correspond to changes in the environment as observed in s. We use Bayesian changepoint detection [3, 15], which can effectively handle noisy time series data. We use the product partition approach [3]. In this approach, the time series data is divided into partitions, such that data in each partition belongs to different probability models. The inference problem thus reduces to identifying the partition given the observation.

We model the agent's state features as time series data and use it to perform change detection. In the example that follows, the state is a tuple $(\langle x, y, \theta \rangle, \delta)$, where the first three terms determine the location and the orientation of the agent and the last term is the value of the distance sensor. Figure 1 shows an example of how the change detection mechanism works using the observations from one particular state in the system. The environment is a 5×5 grid world. The red curve shows the value of δ for a particular location and orientation, $(\langle x, y, \theta \rangle)$. We can see (from the top image) that initially the sensor readings has a mean value of around 5.5 units, in the new environment, the reading reduces, close to zero. This indicates there is an obstacle in the environment at this point. The change detection algorithm correctly shows a high change likelihood at this data point (the blue spike). Change detection calculations happen in each state in the environment.

3.2 Selective exploration

When the agent encounters a changed environment, its past knowledge is insufficient to complete the task. Now the agent selectively explores the sub-space, which captures the changed region of the environment, to learn in the new task. Selective exploration comprises three steps. First, identify and extract the sub-space where the previous knowledge is not applicable. As we show later, the extracted sub-space forms a well-defined MDP that is much smaller than the original task MDP. Next, the agent solves this smaller MDP, to obtain a local policy. Finally, the agent composes the target task policy using the source task policy and the local policy obtained by solving the sub-space MDP.

Sub-space extraction

We exploit the agent's knowledge of the source task to solve the target task. We use a breadth-first like mechanism to extract the sub-space MDP from the parent task. From the state where the environment change is detected, we expand the "horizon" to look ahead n-steps. This enlarged view of the states surrounding the current state forms the sub-space. The expansion follows from the notion of adjacency of states and hence their reachability from the current state. We define these formally below. Once we select the states for selective exploration, we have to determine when the exploration should stop. This is guided by the notion of frontier states that demarcate the current sub-space and rest of the state space in the parent task. Once we have these information we can develop a method to extract a smaller, sub-space MDP from the parent task.

Definition 3.1 (Adjacency). A state s_i is adjacent to a state s_j $(s_i, s_j \in S)$, if there is some action $a \in A$ such that it induces a transition between s_i and s_j, i.e., $T(s_i, a, s_j) > 0$.

Definition 3.2 (Reachability). A state s_j is reachable from state s_i, if s_j is adjacent to s_i or some s_k that is reachable from s_i.

Definition 3.3 (n-step closure). The n-step closure of state s_i, $C^n(s_i)$, is the set of all states reachable from s_i by taking n actions.

Definition 3.4 (Frontier states). The set of states that can be reached from, but outside, the n-step closure on executing one action form the frontier states of the closure. That is, $\mathcal{F}(C^n(s_i)) = \{t \mid \text{for any } s \in C^n(s_i),\ T(s,a,t) > 0,\ \forall a \in A' \wedge \{t\} \cap C^n(s_i) = \varnothing\}$.

Definition 3.5 (Sub-space). Let S be a set of states and $B(S)$ be its boundary states, such that $B(S) \notin S$, and any $s \in S$ is adjacent to some $t \in B(S)$, then (S, G) form a sub-space, iff (i) $S \cap G = \varnothing$, and (ii) $G \subseteq B(S)$

The definition of sub-space above is similar to the one in [5]. The states of the sub-space are obtained from the n-step closure of the particular state under consideration, which is generally where a change is detected in the environment. To define a sub-space MDP, we need to define a set of goal states. Recall that the agent has learned the source task. From this, we know the possible destination states, which are also the destination states of the target task. For instance, for an assistive care robot whose task is to deliver items from specified locations like kitchen to the living room, under the assumptions listed earlier, the landmarks (goal states) remain the same across source and target tasks. The goal states for the sub-space MDP are defined as below.

Lemma 3.6 (Local goal states). *The frontier states form the goal states.*

Proof sketch: Consider the set of states $S' = C^n(s_i)$, which forms the n-step closure of state s_i. The agent's goal is to reach a state that helps circumvent the current change in environment. Any state outside the closure will suffice. Hence the frontier states act as the goal states. □

Lemma 3.7 (Goal preference). *Let, for each $G \in \mathcal{F}$ the reward for reaching G, $r' = r + \Phi(G)$ where Φ is any potential function and r is the reward from the parent task. The reward function is then $R' = \{r'\}$. R' creates a preference order of goal states.*

Proof sketch: By definition, the potential function creates a partial ordering of values in the set R'. Since, the goal states are assigned rewards as per R', there is an ordering over the goal states $G \in \mathcal{F}$. An agent acting greedily will converge to the goal(s) that have the highest reward. Hence a preference order is created over the goal states. □

The states mentioned in Lemma 3.6 are based on the agent's experience in the source task. Lemma 3.6 allows us to construct goal states easily. We use a potential function [30] to create a preference order in the goal states. The potential function is based on a domain dependent heuristic, for instance, it can be defined as the Manhattan distance from goal state in a grid world.

Theorem 3.8 (Sub-space extraction). *There exists $G \in \mathcal{F}$ such that $(C^n(s_i), G)$ form a sub-space.*

Proof: From Definition 3.4, we have that the frontier states and the closure are disjoint sets. Hence, any $G \in \mathcal{F}$ does not belong to the closure $C^n(s_i)$. Now, from Definition 3.5, we have that $(C^n(s_i), G)$ form a sub-space.

Further, using $A' \subseteq A$ as the action set, and R' defined above, for the sub-space, we have a sub-space MDP, $M' = \langle S', A', T', R' \rangle$, where T' is the transition function for this sub-space that must be learned to solve the task. □

In Theorem 3.8 construction, there is one such reward function R' for every sub-space MDP extracted. The advantage of having a reward that is relative in the agent's operational environment is that an absolute frame of reference is not required. In most cases, a suitable reward function can be easily defined based on the current domain of operation.

Sub-space exploration

Earlier we defined how a sub-space MDP is extracted in the target task. Each sub-space MDP is a sub-task in its own right. In our work, we use Rmax exploration to explore in the sub-space extracted. However, one can use any appropriate exploration mechanism in this sub-space MDP. Alternatively, we can also use a different RL algorithm to solve the sub-task MDP.

The extracted sub-space MDP is visible only in the agent space and the environment is oblivious of its presence. Hence the agent has to maintain a mapping between states of the problem and the states in the sub-space MDP. We use a simple look up table mechanism to provide mapping between states in the parent task and those in the sub-space and vice-versa. For complex tasks any other mapping function can be used to map between the parent task and sub-space states. Similarly, all the actions in the parent problem may not be applicable in the sub-space. For instance, if the sub-space extracted is only used in navigating in an environment, other actions like picking up objects or placing them are not relevant. Sub-space exploration yields a policy that circumvents the detected change in the environment. It follows that one such policy is obtained for each sub-space that is extracted.

Policy composition

The result of sub-space exploration and solving the sub-space MDP is a policy that circumvents the change in the environment. To solve the target task, this policy must be used in conjunction with the source task policy. Policy composition can be achieved in various ways. A simple approach is to identify the location of change, and invoke the learned policy in that state. Once the agent is in the goal state local to the extracted sub-space, it can continue to follow the source task policy. The limitation of this approach is that it does not explore the rest of the target task other than the extracted sub-space. Hence the resulting policy may be sub-optimal in the target task. To enforce active exploration of the target task, we use an ε-greedy like approach. With a small probability ε, the agent constructs sub-space MDP even when there are no changes detected in the environment. This sub-space MDP is solved in a similar manner using sub-space exploration. If the extracted sub-space is too small, there is a possibility that the sub-space policy leads to a state that is still blocked, requiring multiple iterations of extracting and solving the sub-space.

Regret calculation

Here we show that the regret an agent accumulates using SEAPoT is dependent only on the step size parameter of the n-step closure.

Definition 3.9 (Diameter). Assuming each action takes unit time, the number of time steps taken to reach state s' from s under policy π in an MDP M with S states and A actions is the Diameter of the MDP [24].

$$D(M) = \max_{s \neq s' \in S,} \min_{\pi : S \to A} \mathbb{E}\left[T(s'|M, \pi, s)\right] \tag{1}$$

From [24], we know that $D = \log_{|A|} |S| - 3$. With respect to the sub-space M', we have $D = \log_{|A'|} |S'| - 3$.

The performance of a policy in a state, s, can be expressed as the expected average reward $\mu^\pi = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}\left[\sum_{t=1}^{T} r^t | s_0 = s\right]$, where $r^t = r(s_t, \pi(s_t))$ is the reward obtained in state s_t following the policy π.

Definition 3.10 (Regret). Regret of the algorithm executing a policy, π, relative to executing the optimal policy, π^*, for T time steps is

$\Delta(s) = T\mu^* - \sum_{t=1}^{T} r^t$ (where μ^* is the optimal average reward) [1].

Theorem 3.11 (Regret as a function of n). *The regret accumulated by the agent following the target task policy obtained using SEAPoT depends only on the step size, n used to construct the sub-space.*

Proof. Consider the target task policy, π', obtained by composing the source task policy and the local policy obtained using selective exploration. The total reward obtained by the agent following π' from state s is: $\mu' = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T} r^t | s_0 = s\right]$. μ' can be expressed as a sum of three parts. Expected reward until change is detected, expected reward following the sub-space policy and the expected reward following the source policy thereafter.

$$\mu' = \lim_{T\to\infty} \frac{1}{T}\left\{\mathbb{E}\left[\sum_{t=1}^{b} r^t\right] + \mathbb{E}\left[\sum_{t=b}^{g} r^t\right] + \mathbb{E}\left[\sum_{t=g}^{T} r^t\right]\right\} \tag{2}$$

From Definition 3.10, we obtain the following regret on using selective exploration instead of acting optimally in the target task. $\Delta = T\mu^* - \mu'$. However, μ^* and μ' differ only in the middle term of (2). Therefore, we can say that the regret obtained by SEAPoT is $T'\mu^* - \mathbb{E}\left[\sum_{t=b}^{g} r^t\right]$. However, $\{b \dots g\}$ are the states of the sub-space S'. Reaching from b to g in the sub-space is equivalent to traversing the diameter of the sub-space MDP, M'. For M', we have $T' = D$. Therefore the regret accrued by a sub-space MDP using SEAPoT is

$$\Delta = D\mu^* - \sum_{t=1}^{D} r^t \tag{3}$$

But, we saw earlier that $D = \left(\log_{|A'|} |S'| - 3\right)$ and $|S'|$ depends on n of the n-step closure (by construction Theorem 3.8). Thus we see that the regret is a function of n, the step size chosen to construct the sub-space. □

4 SEAPOT ALGORITHM

The SEAPoT method is detailed in Algorithm 1. The agent starts by executing the source task policy and performs state transitions until a change is detected in the environment (lines 4–5). The function $\mathbf{f}(s, a)$ indicates executing action a in state s; the agent reaches some state s' on executing $\mathbf{f}$.

In the source task, as the agent learns the policy, it records two things. First, a list of states reachable from every state; second, a set of records of its history. We use the reachable states to construct a n-step reachability closure, $S' = C^n(s_i)$, and the local goal states G'. One step reachable states from a state s are those which can be reached on executing an action a, i.e. $\{s_j \mid T(s_i, a, s_j) > 0\}$.

We use the history recorded to perform change detection in the environment as explained earlier (lines 19–22). When a change in environment is detected, the agent first constructs a sub-space MDP (lines 12–18). Next, the agent explores the target environment until one of the local goal states (i.e, $g' \in G'$) is reached (lines 9–11). The intuition here is to quickly reach a state from where the source policy can be followed. On reaching g', the agent derives the target task policy by composing from the source task policy and the policy obtained from the sub-space MDP. The process repeats until the agent reaches the goal in the target task or it encounters another change in the environment. We notice two things from the algorithm. (i) The exploration is limited to a small number of steps (till reaching $g' \in G'$) (ii) The number of times exploration is performed is minimal.

5 MEASURING TASK DIFFERENCE

To prevent or minimize negative transfer, i.e., transferring knowledge across dissimilar tasks that leads to degradation in the learning performance, we need to determine if the source and target tasks are "similar". It is difficult to determine task similarity a priori, unless the transition functions, value functions and/or reward functions for both tasks are available. Also, similarity metric is dependent on the transfer mechanism considered and it is impossible to define a single metric to measure task similarity for all transfer mechanisms [8].

Prior efforts in determining task similarity require pre-defined task models. Two common metrics are used to determine the task similarity, viz., the Kantorovich distance metric [18, 32] and Bisimulation metric [19]. Bisimulation metric is an exact metric. Unless the states transitions and rewards are equivalent, the metric reports dissimilarity in the tasks. On the other hand, the worst case computational complexity to determine the Kantorovich distance is $O\left(|A||S_1|^2|S_2|^2|S_1 + S_2|\log(S_1 + S_2)\lceil\frac{\ln\eta}{\ln c}\rceil\right)$. Ignoring the log terms, this is quartic in the size of the state space (if $|S_1| \approx |S_2|$).

We define a new, light weight, metric based on the Jensen-Shannon distance [14] (JSD) to compute task similarity in the problems that share the same state-actions. Jensen-Shannon distance is defined as the square root of the Jenson-Shannon divergence, D_{JS}. JSD is computed as shown in Equation (4).

$$\begin{aligned} D_{JS}(p, q) &= \frac{1}{2}D_{KL}(p, m) + \frac{1}{2}D_{KL}(q, m) \\ JSD &= \sqrt{D_{JS}} \end{aligned} \tag{4}$$

where, $m = \frac{p+q}{2}$, D_{KL} is the Kullback-Leibler divergence, and p and q are any two probability distributions. The computation complexity of JSD is linear in the number of elements in the two distributions [23].

The task difference ($\Delta_{S,T}$) is calculated as follows. Since we are dealing with the shared state-action space, we calculate the bin-bin distance, JSD, among the corresponding state-action transition distributions of the two tasks. Hence, in (4), p and q are the transition distributions of the corresponding state-action pairs in the source and target tasks. The JSD calculated between each corresponding state-action pair is passed through a step function ($\mathbb{I}$) to determine its contribution to the task difference. The task difference is then the summation of $\mathbb{I}$ over all the state-action pairs as shown in (5).

$$\Delta_{S,T} = \sum_{(s,a)} \mathbb{I}(JSD) \tag{5}$$

where, $\mathbb{I}(\cdot)$ is a step function. For example, we can say, all state-action pairs that have $JSD > 0.5$ contribute to the task difference. Hence,

$$\mathbb{I}(x) = \begin{cases} 1 & \text{if } x > 0.5 \\ 0 & \text{otherwise.} \end{cases}$$

```
Algo SEAPoT:
while Target task NOT solved:
    With probability $\epsilon$ explore target;
    while NOT ChangeDetect():
        $s' \leftarrow \mathbf{f}(s, \pi^S(s))$
    Extract sub-space MDP();
    Explore sub-space MDP( );
    Compose policy $\pi^T$
Proc Explore sub-space MDP:
    Input: $subSpaceMDP$
    Explore using R-MAX until $g' \in G'$ reached ;
Proc Extract sub-space MDP:
    // identify $n$-step closure
    $S' \leftarrow C^n(s)$ // According to Defn 3.3
    // identify the local goal
    $G' \leftarrow \mathcal{F}(S')$ // According to Lem 3.6
    Define $R'$ // According to Lem 3.7
    Construct $M'$ // According to Thm 3.8
Proc ChangeDetect:
 Input: Time series data D
if Change detected:
     Return True
```

Algorithm 1: SEAPoT Algorithm

Like in previous work, we run a few episodes of the target task to obtain an initial estimate of the transition probabilities and use this to determine the task similarity. In the experiments section we report the improvement in learning (in terms of accumulated rewards) for each similarity measure obtained.

Intuitively, the task difference metric $\Delta_{S,T}$ identifies the number of locations where the environment has changed from the source task to the target, where the source knowledge is no longer applicable in the target.

6 RELATED WORK

Many existing policy reuse methods, based on model-free Q-learning can be categorized into two main categories: (i) developing an inter-task mapping between source and target tasks, and (ii) developing a policy library which can be reused in a new target task.

Developing an inter-task mapping is a common approach to policy transfer between the source and target tasks [17, 33, 36]. The mapping can be obtained in different ways. For example, (a) mapping from the semantics of the features and actions is used to derive the target policy given the domain description and the source policy [17]; (b) neural networks, with input nodes describing the current state and the output node denoting the action selected, trained on the source tasks are used with appropriate state variable mapping to derive the policy in the target task [36]; and (c) mapping between the state-action pairs of the source and target tasks is used to transfer the source policy as option (or macro-action) in the target task [33]. Given the similarity assumption between the source and target tasks, we assume the action mapping to be one-to-one from source to target; there is no state mapping. Our approach, however, allows and admits different transition functions in the source and target tasks.

Policy library methods treat the problem of policy transfer like a multi-armed bandit problem. The solution mechanism involves identifying the best policy to reuse from a set of learned policies without performing any shaping or inter-task mapping [9, 16]. Our work is similar to these methods, but differ in following ways. First, the existing methods work with the constraint that the domain (defined as a tuple consisting of the states, actions and transition functions) remains the same and each task is differentiated by the reward functions. In our work, the transition functions involved in the tasks are different; the reward functions may or may not be the same across source and target tasks. Second, the existing methods adopt probabilistic policy reuse mechanisms. When the state-action spaces are similar in the source and target tasks, definitive policy reuse and selective exploration perform better than probabilistic policy reuse. Our approach can be extended with a probabilistic reuse mechanism in future.

In RL, the STAR-MAX [29] algorithm uses domain knowledge to reduce the sample complexity of R-MAX by preventing exploration in irrelevant states. The algorithm, however, requires the exploration envelope (a set of states to target exploration) and a recovery rule (a policy to return the agent to the exploration envelope) to be provided. In transfer learning, it is impossible to provide such subsets of the state space for each task. We use the state reachability information from the source task to selectively explore around "obstacles" and quickly reach a state from which the original source policy can be followed.

Model minimization techniques have been used to solve MDP by aggregating states or abstracting states in a large MDP [11, 20, 25]. In RL, hierarchical decomposition has been used in online planning [2]. However, these techniques do not extract a sub-space in the given problem. Barry et.al., [4] propose inducing a hierarchy in large state space MDP. Our method of extracting sub-space builds on the notion of adjacency and reachability defined in a similar manner.

Transferring knowledge from the source to target task can be effected by other means like value function or trajectory transfer. Information transferred by policy reuse is much lesser as compared to other transfer methods. For instance, given the value function or state-action-reward trajectories one can extract the policy but it is generally not true the other way round. In different settings, different mechanisms can be more effective than others. In general, policy transfer mechanisms are space efficient owing to the design. The transfer agent has to maintain only the source task policies, instead of maintaining information about the transition model of the entire state space or the history of state-action trajectories. To the best of our knowledge no previous work, using any transfer mechanism, has attempted a selective exploration approach as we propose.

7 EXPERIMENTS

In our experiments, we mimic the real-world setting for a household robot that operates in a set of co-designed/co-developed homes, (possibly catering to a specific need such as housing patients with with limited mobility). This set up is common, for example, in the public housing estates in Singapore. While the layout of the houses is fixed, the environments differ in the placement of furniture and other artifacts in the home. We evaluate the performance and efficacy of SEAPoT in two sets of problems—navigation in a stochastic shortest path setting and a pick-and-place task. We compare the performance of SEAPoT with three state-of-the-art policy transfer methods, Bayesian policy reuse [31], probabilistic policy reuse [16], and policy reuse with reward shaping [7].

One of the most basic tasks for any assistive robot is navigation within the environment. We model the environment as a discrete two dimensional space. We explore two scenarios in our experiments. The first is a simple navigate to target setting, where the robot starts anywhere in the environment and reaches a target location. This is similar to a robot reaching the kitchen when summoned by the user. In the second scenario, the robot is tasked to pick up an object from one of the pre-determined locations and place it in a different location. This scenario encompasses many household tasks. Some examples include fetching medicine from the bedroom to living room where the user is sitting, clearing used-cutlery from the dining table and moving them to the dish washer, etc. This setting is similar to the benchmark taxi problem described by [12].

We used environments with varying sizes to demonstrate the scalability of our approach[1]. Our evaluations were performed on three environments: taxi like environment [12] that has 2000 states; a simulated household robot (similar to [13]) with 8000 states and Microsoft Malmo platform [26] based on the popular game Minecraft, with about 63000 states.

The taxi problem comprises a grid with numerous landmarks. The state is represented as the vector $\langle x, y, \theta, p, d, \delta \rangle$. Where (x, y, θ) represent the agent's location and orientation; p and d represent the passenger location and destination respectively. δ is the reading from the distance sensor of the agent in each location in the grid. The state-space size of this environment (calculated as $|x| \times |y| \times |\theta| \times |p| \times |d|$; δ is omitted as it is not used in the planning) can be varied using different values of x (rows), y (columns) and the number of landmarks in the grid. A state in the Minecraft environment is represented similarly, with the addition of a vector of binary values to indicate the resources the agent is currently holding and the agents view point. In each of the test beds the agent can perform a total of five actions: `move forward`, `turn left`, `turn right`, `pick up object` and `put down object`.

In all the experiments we use R-MAX as the underlying exploration mechanism and the standard value iteration to solve the MDP. We set the exploration threshold of R-MAX to one, to prevent unnecessary exploration. There is a penalty of -1 for each step taken, and reward of +20 for successful completion of the task.

[1]Our experiments are comparable in size and complexity to those reported in other state of the art work. For example, the complex Mario domain [27] used in reward shaping transfer work has 7072 discrete states [7].

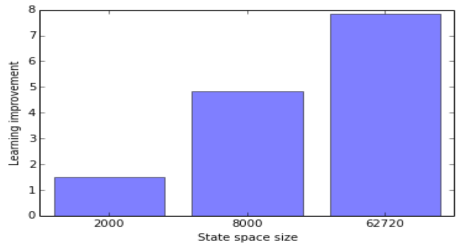


Figure 2: Learning improvement

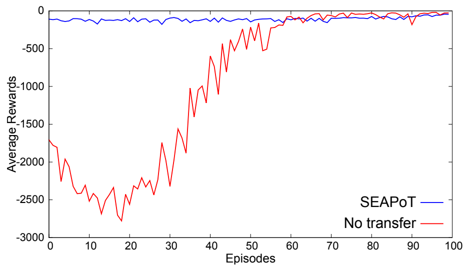


Figure 3: SEAPoT versus no transfer

For pick-and-place tasks, an invalid pick up or put down attracts a penalty of -10.

The actions have a probabilistic effect and the agent transits to the intended state 80% of the time. It has a 20% chance of turning either left or right. Each experiment run consisted of 100 episodes and was repeated twenty times and the rewards were averaged over these runs. Since the outcomes are similar for the pick-and-place tasks across the environments, we only report the results of the experiments on the simulated home environment. Learning time for the source task is considered as sunk cost [35].

Performance of SEAPoT

We define learning improvement as the ratio of the accumulated rewards obtained without transfer and accumulated rewards using SEAPoT. The learning improvement calculated in all the environments is shown in Figure 2. Due to the way in which we extract the sub-space and limit the learning in the target task, the improvement in the accumulated rewards is apparent in problems with larger state space than in the smaller ones, proving that SEAPoT performs better in problems with larger state spaces. The improvement in jump start is shown in Figure 3. Notice the improvement in the initial rewards obtained by using SEAPoT as compared to learning the target task without transfer. In both cases, the learning time of the source task is not considered in reporting, like any other previous transfer learning works.

A flexible policy transfer method

In the next experiment we intend to demonstrate two-fold advantage of our work. First, the policy transfer method is designed to be flexible—we can use any exploration strategy or learning algorithm. To demonstrate this, we incorporate two different exploration mechanisms. Second, in the simplest terms, our approach involves identifying the change and circumventing the same. One

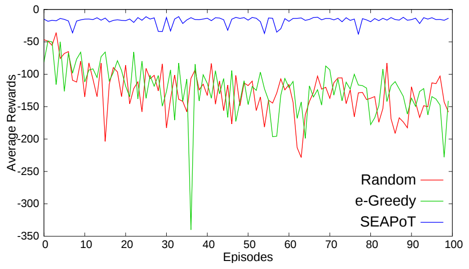


Figure 4: Comparison of different exploration strategies

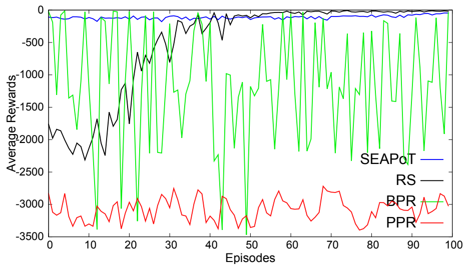


Figure 5: Comparison between SEAPoT, PPR, RS and BPR

can argue that any simplistic exploration strategy should suffice. To show the advantage of SEAPoT over simple exploration strategies, we compare our work with two baseline exploration mechanisms (i) random exploration (ii) ϵ-greedy like exploration. In all cases, we use the same source policy as the transferred knowledge. The first approach is a naive one; when the agent detects a change in the environment, it takes a set of random actions hoping to reach a state that is known. In the second case, the agent takes a random action with a small probability ϵ (set to 10% in this case), when a change is detected. Otherwise, it follows the source task policy. In contrast, SEAPoT extracts the sub-space and performs minimal exploration in the extracted sub-space, learning the dynamics of the new environment. The average rewards obtained for each exploration strategy is shown in Figure 4. The baseline methods fail to converge to the optimal policy, whereas SEAPoT performs better.

SEAPoT for Policy Transfer

Next, we compare our work with three state-of-the-art policy reuse methods [7, 16, 31] and the classical Dyna-Q [34], and analyze the impact of policy reuse with selective exploration in the target environments. Since the experimental environments are different in each of the earlier works, we evaluated all the algorithms using the experimental setup described earlier.

We compare the methods based on the average reward obtained. Dyna-Q does not converge to the optimal policy even after hundred episodes owing to the large state-space size of the problem(s). Hence, we do not include it in the comparison presented in Figure 5. Notice the improvement in the average rewards of SEAPoT over both probabilistic policy reuse and policy transfer using reward shaping. We attribute the performance improvement to the following reasons: (i) The agent reuses its behavioral knowledge learned in the source to the maximum extent in the target task; (ii) there is minimal exploration in the target environment. Probabilistic policy reuse (PPR), irrespective of the current situation (possibly of being stuck behind an obstacle), applies the same action as the source policy, if the outcome of the decision is to use the source policy. Hence the agent draws negative rewards. Bayesian policy reuse (BPR) mechanism works well when the set of tasks are known beforehand. When a new task is presented, it selects a policy from the library and reuses it.

Since the task presented to BPR is not among the ones that are learned earlier, it doesn't converge in our experiments. The reward shaping (RS) agent, on the other hand, has to learn the target task, albeit with help from the shaping signal that is obtained from the source task policy. Hence, even though it converges to a better policy (notice that SEAPoT converges to a similar policy) the jump start is lost.

Effect of Task Difference on Performance

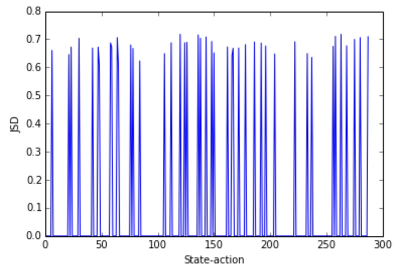
Earlier we described a new metric to determine the difference between the source and target tasks. We hypothesize that the performance of SEAPoT degrades when the source and the target tasks are drastically different. In this experiment we use a simple navigation task in the small MAXQ taxi-like environment to verify the hypothesis. Without loss of generality, let us call the tasks as $T1$, $T2$, $T3$ and $T4$. We use $T1$ as the source task and the others as the target. We plot the differences between each pair of tasks in Figure 6. The spikes indicate the state-action combinations where the two tasks differ[2]; hence, lesser number of spikes mean the tasks are closer to each other. From Figure 6, we can see that $T4$ is closer to $T1$ than $T2$ and we expect transferred knowledge to have a positive impact on learning $T4$. To show the performance difference of SEAPoT at different degrees of similarity, we plot the average rewards of two tasks, $T3$ and $T4$ in Figure 7. As one can expect the performance degradation is more pronounced in the case of $T2$ and we don't show it to maintain the aesthetics of the plot. Next, we plot the accumulated rewards for each of the target tasks at the end of the learning in Figure 8. As hypothesized, $T2$ suffers from negative transfer whereas $T4$ does better with transfer.

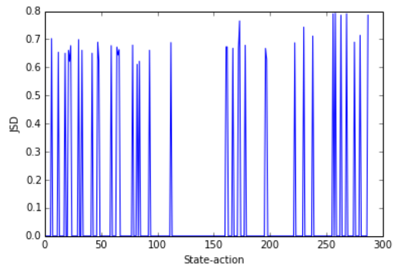
One can use these observations as a ballpark estimate to set the threshold and to determine when the framework should discard the source policy and learn the target task afresh. We leave determining the theoretical threshold value to future work.

8 DISCUSSION AND CONCLUSION

We presented a new sub-space extraction mechanism that aids selective exploration for policy transfer in RL. The proposed SEAPoT algorithm explores in the resulting sub-space, thereby reducing exploration required in the target task. Though our assumption on similar source and target tasks sounds restrictive, it covers many real world use cases. We showed that our algorithm works well in problems with large state spaces; the framework is flexible and allows incorporation of different exploration mechanisms and learning algorithms. Our work is a first attempt towards reuse of source task policy by exploiting online, local information in the target tasks to adapt to the new setting.

[2]Though we use factored notation in our work, for ease of understanding, the x-axis in the graph is the enumerated state-action space


(a) $\Delta_{T1,T2} = 44$


(b) $\Delta_{T1,T3} = 36$

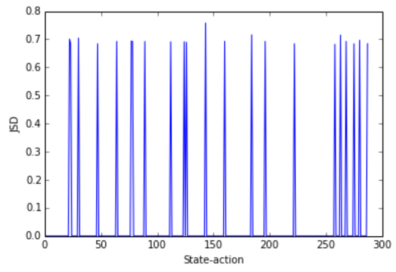

(c) $\Delta_{T1,T4} = 22$

Figure 6: Difference between source and target tasks (each Δ indicates the number of locations where the target task differs from the source task T1)

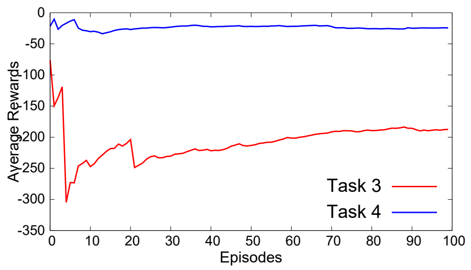


Figure 7: Performance of SEAPoT for two target tasks, $T3$ and $T4$

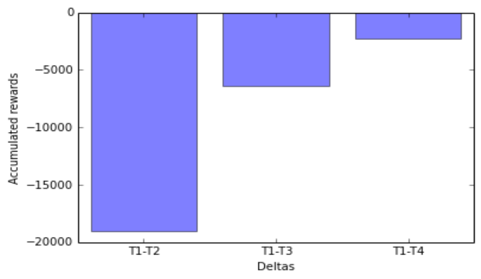


Figure 8: Performance of SEAPoT at different Δ's

Selective exploration may sometimes lead to costly additional exploration steps. We can alleviate this problem by adaptively increasing the size, n, of the n-step closure. In this work, the value of n was determined empirically. We will provide a theoretical evaluation to determine the optimal value of n in a subsequent paper. Active exploration of the target task is still minimal in the proposed setup and will be addressed in future work. Using simulation for look-ahead to identify tasks structure and partial policy reuse in the target task can help relax the similar tasks assumption. In model-based methods, the learned environment models can be used to encode valuable information that can be transferred to the target task. We intend to extend our work to exploit the information contained in the learned models to generate better sub-spaces and reducing the sample complexity of the target task learning. Finally, another planned extension is to use the task similarity information to make the key decision—to transfer source knowledge or learn the task afresh. This decision depends on the level of performance loss acceptable due to the transfer, which in turn is specific to the domain of interest.

ACKNOWLEDGMENTS

This research was partially supported by Academic Research Grants: MOE2016-T2-2-068 and T1-251RES1827 from the Ministry of Education in Singapore.

REFERENCES

[1] Mohammad Gheshlaghi Azar, Alessandro Lazaric, and Emma Brunskill. 2013. Regret Bounds for Reinforcement Learning with Policy Advice. In *Machine Learning and Knowledge Discovery in Databases*. Lecture Notes in Computer Science, Vol. 8188. Springer Berlin Heidelberg, 97–112.

[2] Aijun Bai, Feng Wu, and Xiaoping Chen. 2015. Online planning for large markov decision processes with hierarchical decomposition. *ACM Transactions on Intelligent Systems and Technology (TIST)* 6, 4 (2015), 45.

[3] Daniel Barry and John A Hartigan. 1993. A Bayesian analysis for change point problems. *J. Amer. Statist. Assoc.* 88, 421 (1993), 309–319.

[4] Jennifer L. Barry, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2011. DetH*: Approximate Hierarchical Solution of Large Markov Decision Processes. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, Vol. 3. AAAI Press, 1928–1935. doi: 10.5591/978-1-57735-516-8/IJCAI11-323.

[5] Michael Bowling and Manuela Veloso. 1998. Reusing Learned Policies Between Similar Problems. In *Proceedings of the AI*IA-98 Workshop on New Trends in Robotics*. Padua, Italy.

[6] Ronen I Brafman and Moshe Tennenholtz. 2003. R-max—A general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research* 3 (2003), 213–231.

[7] 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 (AAMAS-15)*. 181–188.

[8] James L Carroll and Kevin Seppi. 2005. Task similarity measures for transfer in reinforcement learning task libraries. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN-05)*, Vol. 2. IEEE, 803–808.

[9] Yann Chevaleyre and AydanoMachado Pamponet. 2012. Adaptive Probabilistic Policy Reuse. In *Neural Information Processing*. Lecture Notes in Computer Science, Vol. 7665. Springer Berlin Heidelberg, 603–611.

[10] 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.

[11] Luca De Alfaro and Pritam Roy. 2007. Magnifying-lens abstraction for Markov Decision Processes. In *Computer Aided Verification*. Springer, 325–338.

[12] Thomas G Dietterich. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.(JAIR)* 13 (2000), 227–303.

[13] Carlos Diuk, Andre Cohen, and Michael L. Littman. 2008. An Object-oriented Representation for Efficient Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*. ACM, New York, NY, USA, 240–247.

[14] Dominik Maria Endres and Johannes E Schindelin. 2003. A new metric for probability distributions. *IEEE Transactions on Information Theory* 49, 7 (2003),

1858–1860.

[15] Chandra Erdman, John W Emerson, et al. 2007. bcp: An R package for performing a Bayesian analysis of change point problems. *Journal of Statistical Software* 23, 3 (2007), 1–13.

[16] Fernando Fernández, Javier García, and Manuela Veloso. 2010. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems* 58, 7 (2010), 866–871.

[17] Fernando Fernández and Manuela Veloso. 2006. Policy reuse for transfer learning across tasks with different state and action spaces. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*.

[18] Norm Ferns, Prakash Panangaden, and Doina Precup. 2004. Metrics for finite Markov decision processes. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI-04)*. AUAI Press, 162–169.

[19] Norm Ferns, Doina Precup, and Sophia Knight. 2014. Bisimulation for markov decision processes through families of functional expressions. In *Horizons of the Mind. A Tribute to Prakash Panangaden*, Franck van Breugel, Elham Kashefi, Catuscia Palamidessi, and Jan Rutten (Eds.). Springer, 319–342.

[20] Robert Givan, Thomas Dean, and Matthew Greig. 2003. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence* 147, 1 (2003), 163–223.

[21] Carlos Guestrin, Relu Patrascu, and Dale Schuurmans. 2002. Algorithm-directed Exploration for Model-based Reinforcement Learning in Factored MDPs. In *Proceedings of the 19th International Conference on Machine Learning (ICML-02)*. 235–242.

[22] Majd Hawasly and Subramanian Ramamoorthy. 2013. Lifelong learning of structure in the space of policies. In *AAAI Spring Symposium: Lifelong Machine Learning*.

[23] Yuma Iwasaki, A Gilad Kusne, and Ichiro Takeuchi. 2017. Comparison of dissimilarity measures for cluster analysis of X-ray diffraction data from combinatorial libraries. *npj Computational Materials* 3, 1 (2017), 4.

[24] Thomas Jaksch, Ronald Ortner, and Peter Auer. 2010. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research* 11, Apr (2010), 1563–1600.

[25] Qing-Shan Jia. 2011. On state aggregation to approximate complex value functions in large-scale Markov decision processes. *Automatic Control, IEEE Transactions on* 56, 2 (2011), 333–344.

[26] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. 2016. The Malmo platform for artificial intelligence experimentation. In *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 4246–4247.

[27] Sergey Karakovskiy and Julian Togelius. 2012. The Mario AI benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 55–67.

[28] Alessandro Lazaric. 2012. Transfer in reinforcement learning: A framework and a survey. In *Reinforcement Learning*. Springer, 143–173.

[29] Timothy A. Mann and Yoonsuck Choe. 2011. Scaling up reinforcement learning through targeted exploration. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)*. 435–440.

[30] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, Vol. 99. 278–287.

[31] Benjamin Rosman, Majd Hawasly, and Subramanian Ramamoorthy. 2016. Bayesian policy reuse. *Machine Learning* (2016), 1–29. doi: 10.1007/s10994-016-5547-y.

[32] Jinhua Song, Yang Gao, Hao Wang, and Bo An. 2016. Measuring the distance between finite markov decision processes. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems (AAMAS-16)*. International Foundation for Autonomous Agents and Multiagent Systems, 468–476.

[33] Vishal Soni and Satinder Singh. 2006. Using Homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of the 21st national conference on Artificial intelligence-Volume 1*. AAAI Press, 494–499.

[34] Richard S. Sutton. 1990. Integrated Architecture for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Proceedings of the Seventh International Conference on Machine Learning (ICML-1990)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 216–224.

[35] Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research* 10 (2009), 1633–1685.

[36] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. 2007. Transfer via Inter-task Mappings in Policy Search Reinforcement Learning. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-07)*. Article 37, 8 pages. doi: 10.1145/1329125.1329170.