Probabilistic Policy Reuse in a Reinforcement Learning Agent

Fernando Fernández
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
fernando@cs.cmu.edu

Manuela Veloso
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
veloso@cs.cmu.edu

ABSTRACT

We contribute Policy Reuse as a technique to improve a reinforcement learning agent with guidance from past learned similar policies. Our method relies on using the past policies as a probabilistic bias where the learning agent faces three choices: the exploitation of the ongoing learned policy, the exploration of random unexplored actions, and the exploitation of past policies. We introduce the algorithm and its major components: an exploration strategy to include the new reuse bias, and a similarity function to estimate the similarity of past policies with respect to a new one. We provide empirical results demonstrating that Policy Reuse improves the learning performance over different strategies that learn without reuse. Interestingly and almost as a side effect, Policy Reuse also identifies classes of similar policies revealing a basis of *core* policies of the domain. We demonstrate that such a basis can be built incrementally, contributing the learning of the structure of a domain.

1. INTRODUCTION

We introduce Policy Reuse as a technique for reinforcement learning guided by past policies. Policy Reuse balances among exploitation of the ongoing learned policy, exploration of random actions, and exploration of the past policies. The exploration versus exploitation tradeoff defines whether to explore unseen parts of the space or to exploit the knowledge already acquired during the learning process. Several strategies, such as ϵ-greedy, Boltzmann or directed exploration [16], have been developed to balance this tradeoff. However these strategies only exploit knowledge obtained in the current learning process.

Several methods aim at improving learning by introducing additional knowledge into the exploration choices. Advice rules [7] define the actions to be preferred in different sets of states. The user provides the advice rules. Different knowledge sources can be used, for example through learning policies by imitation [10]. Other algorithms introduce previous knowledge through macro-actions or sub-policies. Macro-actions can be used to learn new action policies in Semi-Markov Decision Processes (SMDPs) [18]. Options can also be used in SMDPs by learning their behavior on line [13].

Hierarchical RL uses different abstraction levels to organize subtasks [3], and some approaches are able to learn such a hierarchy [6]. The methods for learning hierarchies or options capture the structure of the domain. Some related algorithms are SKILL [17], which discovers partially defined policies that arise in the context of multiple tasks in the same domain, and L-Cut, which discovers subgoals and corresponding sub-policies [12]. Sub-policies can suboptimally solve a task with computable bounds [1].

Transfer learning, as knowledge reuse across different learning task, can be performed by initializing the Q-values of a new episode with previously learned Q-values [2, 8]. However if the source and target tasks are very different, transfer learning requires expert knowledge to decide on the feasibility of the transfer, and on the mapping between actions and states from the source and target tasks [14, 15]. Value function transfer is an alternative but it is restricted to previous learning processes performed also through a value function.

In this paper we contribute Policy Reuse, a reinforcement learning method in which learned policies are saved and reused for similar tasks. The main algorithm of Policy Reuse is the PRQ-Learning algorithm, a method to probabilistically reuse a set of past policies that solve different tasks within the same domain. The PRQ-Learning algorithm has two main ideas. First, we introduce the π-reuse exploration strategy, which is able to probabilistically bias the exploration to include a given predefined past policy; and second, we introduce the similarity function that allows the estimation of the usefulness of past policies with respect to learning a new task.

We further present the PLPR algorithm, an incremental method to build a library of policies. When solving a new problem by policy reuse, the PLPR algorithm determines how different the learned policy is from the past policies as a function of the effectiveness of the reuse. If the past and new policies are "sufficiently" different, PLPR decides to add the new policy to the library of policies. Otherwise, it does not. PLPR is therefore capable of identifying a set of "core" policies that need to be saved to solve any new task in the domain within a threshold of similarity, δ. Given δ, our algorithm identifies a set of "δ-core-policies," as the basis or learned structure of the domain. Thus our method to build the Policy Library has a novel "side-effect" in terms of learning the structure of the domain, i.e., the basis or the core policies of the domain.

In summary, Policy Reuse contributes to the overall goal of a lifelong reinforcement learning agent, as (i) it provides a mechanism to reuse past policies; (ii) it incrementally builds a policy library; and (iii) it learns an abstract domain structure in terms of core policies of the domain.

The paper is organized as follows. Section 2 introduces Policy Reuse for a reinforcement learning agent. Section 3 defines the π-reuse exploration strategy and presents experiment results in a large robot navigation domain. Section 4 formalizes the similarity metric among policies and introduces the PRQ-Learning algorithm. Section 5 presents the PLPR algorithm. Lastly, Section 6 draws conclusions.

2. POLICY REUSE IN REINFORCEMENT LEARNING

Reinforcement Learning problems are typically formalized using Markov Decision Processes (MDPs). An MDP is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T}$ is a stochastic state transition function, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re$, and $\mathcal{R}$ is a stochastic reward function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$. RL assumes that $\mathcal{T}$ and $\mathcal{R}$ are unknown.

We focus in RL domains where different *tasks* can be solved. The MDP's formalism is not expressive enough to represent all the concepts involved in knowledge transfer [11], so we define domain and task separately to handle different tasks executed in the same domain. We introduce a task as a specific reward function, while the other concepts, $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{T}$ stay constant for all the tasks in the same domain. We characterize a domain, $\mathcal{D}$, as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T} >$. We define a task, Ω, as a tuple $< \mathcal{D}, \mathcal{R}_\Omega >$, where $\mathcal{D}$ is a domain as previously defined, and $\mathcal{R}_\Omega$ is the stochastic and unknown reward function.

Definition 1. *A Domain $\mathcal{D}$ is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T} >$, where $\mathcal{S}$ is the set of all states; $\mathcal{A}$ is the set of all actions; and $\mathcal{T}$ is a state transition function, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re$.*

Definition 2. *A task Ω is a tuple $< \mathcal{D}, \mathcal{R}_\Omega >$, where $\mathcal{D}$ is a domain; and $\mathcal{R}_\Omega$ is the reward function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$.*

We assume that we are solving episodic tasks with absorbing goal states. Thus if s_i is a goal state, the probability of transitioning to the same state is 1 ($\mathcal{T}(s_i, a, s_i) = 1$), hence the transition probability to a different state is 0 ($\mathcal{T}(s_i, a, s_j) = 0$ for $s_i \neq s_j$), and the immediate reward is 0 ($\mathcal{R}(s_i, a) = 0$, for all $a \in \mathcal{A}$).

A trial or episode starts by locating the learning agent in a random position in the environment. Each episode finishes when the agent reaches a goal state or when it executes a maximum number of steps, H. The agent's goal is to maximize the expected average reinforcement per episode, W, as defined in equation 1:

$$W = \frac{1}{K} \sum_{k=0}^{K} \sum_{h=0}^{H} \gamma^h r_{k,h} \tag{1}$$

where γ ($0 \leq \gamma \leq 1$) reduces the importance of future rewards, and $r_{k,h}$ defines the immediate reward obtained in the step h of the episode k, in a total of K episodes.

An action policy, Π, is a function $\Pi : \mathcal{S} \rightarrow \mathcal{A}$ that defines how the agent behaves. If the action policy was created to solve a defined task, Ω, we call that action policy Π_Ω. The gain, or average expected reward, received when executing an action policy Π in the task Ω is called W_Ω^Π. Lastly, an optimal action policy for solving the task Ω is called Π_Ω^*. The action policy Π_Ω^* is optimal if $W_\Omega^{\Pi_\Omega^*} \geq W_\Omega^\Pi$, for all policy Π in the space of all possible policies when $K \rightarrow \infty$. Action policies can be represented using the action-value function, $Q^\Pi(s, a)$, which defines for each state $s \in \mathcal{S}$, $a \in \mathcal{A}$, the expected reward that will be obtained if the agent starts to act from s, executing a, and after it follows the policy Π. So, the RL problem is mapped to learning the function $Q^\Pi(s, a)$ that maximizes the expected gain. The learning can be performed using different algorithms, such as Q-Learning [19].

The goal of Policy Reuse is to use different policies, which solve different tasks, to bias the exploration process of the learning of the action policy of another similar task in the same domain. We call Policy Library to the set of past policies, as defined next.

Definition 3. *A Policy Library, L, is a set of n policies $\{\Pi_1, \ldots, \Pi_n\}$. Each policy $\Pi_i \in L$ solves a task $\Omega_i =< \mathcal{D}, \mathcal{R}_{\Omega_i} >$, i.e., each policy solves a task in the same domain.*

The previous definition does not restrict the characteristics of the tasks (they may be repeated), nor the characteristics of the policies (they may be sub-optimal). The scope of Policy Reuse is summarized as: we want to solve the task Ω, i.e., learn Π_Ω^*; we have previously solved the set of tasks $\{\Omega_1, \ldots, \Omega_n\}$ with n policies stored as a Policy Library, $L = \{\Pi_1, \ldots, \Pi_n\}$; how can we use the policy library, L, to learn the new policy, Π_Ω^*?

Policy Reuse answers this question by adding the past policies into a learning episode as an probabilistic exploration bias. We define an exploration strategy able to bias the exploration process towards the policies of the Policy Library, and a method to estimate the utility of reusing each of them and to decide whether to reuse them or not. Furthermore, Policy Reuse provides an efficient method to construct the Policy Library. We now detail the Policy Reuse approach.

3. REUSING A PAST POLICY

We now describe how one policy that solves a particular task can be used to bias the learning of an action policy for a similar task. Formally, we want to solve a new task Ω, i.e., learn Π_Ω^*. We assume that we are given a Policy Library, say $L = \{\Pi_1, \ldots, \Pi_n\}$ and that there is an oracle that returns Π_{past} as the most similar past policy to Π_Ω^*. We present how our algorithm reuses the policy Π_{past}.

Section 4 later defines a similarity metric between policies for different tasks, and introduces how to automatically estimate the policy to reuse.

3.1 The π-reuse Exploration Strategy

The π-reuse strategy is an exploration strategy able to bias a new learning process with a past policy. Let Π_{past} be the past policy to reuse and Π_{new} the new policy to be learned. We assume that we are using a direct RL method to learn the action policy, so we are learning the related Q function. Any RL algorithm can be used to learn the Q function, with the only requirement that it can learn off-policy, i.e., it can learn a policy while executing a different one, as Q-Learning does [19].

The goal of π-reuse is to balance random exploration, exploitation of the past policy, and exploitation of the new policy, as represented in Equation 2.

$$a = \begin{cases} \Pi_{past}(s) & \text{w/prob. } \psi \\ \epsilon - greedy(\Pi_{new}(s)) & \text{w/prob. } (1-\psi) \end{cases} \qquad (2)$$

The π-reuse strategy follows the past policy with probability ψ, and it exploits the new policy with probability of $1-\psi$. As random exploration is always required, it exploits the new policy with an ϵ-greedy strategy.

Table 1 shows a procedure describing the π-reuse strategy integrated with the Q-Learning algorithm. The procedure gets as an input the past policy Π_{past}, the number of episodes K, the maximum number of steps per episode H, and the ψ parameter. An additional υ parameter is added to decay the value of ψ in each step of the learning episode. The procedure outputs the Q function, the policy, and the average gain obtained in the execution, W, which will play an important role in similarity assessment, as the next sections present. The variable ψ_h keeps the value of $\upsilon^h\psi$ in each step of each episode.

```
$\pi$-reuse ($\Pi_{past}, K, H, \psi, \upsilon$).
Initialize $Q^{\Pi_{new}}(s,a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
For $k = 0$ to $K-1$
    Set the initial state, $s$, randomly.
    Set $\psi_1 \leftarrow \psi$
    for $h = 1$ to $H$
        With a probability of $\psi_h$, $a = \Pi_{past}(s)$
        With a probability of $1-\psi_h$, $a = \epsilon$-greedy($\Pi_{new}(s)$)
        Receive the next state $s'$, and reward, $r_{k,h}$
        Update $Q^{\Pi_{new}}(s,a)$, and therefore, $\Pi_{new}$:
            $Q^{\Pi_{new}}(s,a) \leftarrow (1-\alpha)Q(s,a)^{\Pi_{new}} +$
                $\alpha[r + \gamma \max_{a'} Q^{\Pi_{new}}(s',a')]$
        Set $\psi_{h+1} \leftarrow \psi_h \upsilon$
        Set $s \leftarrow s'$
$W = \frac{1}{K}\sum_{k=0}^{K}\sum_{h=0}^{H}\gamma^h r_{k,h}$
Return $W$, $Q^{\Pi_{new}}(s,a)$ and $\Pi_{new}$
```

Table 1: π-reuse Exploration Strategy.

3.2 Experiments

We describe the experiments performed to demonstrate the usefulness of the π-reuse exploration strategy. We first describe the experimental domain used.

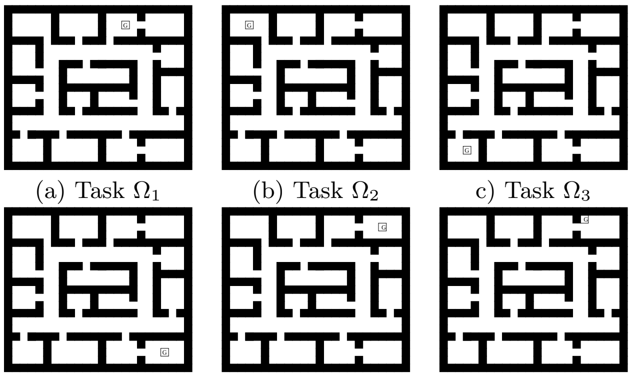
3.2.1 Robot Navigation Domain

We use a grid-based robot navigational domain (see Figure 1) with multiple rooms. The environment is represented by walls, free positions and goal areas, all of them of size 1×1. The whole domain is $N \times M$ (24×21 in our case). The actions that the robot can execute are "North," "East," "South," and "West", all of size one. The final position after executing an action is noised by adding to the new position a random value that follows a uniform distribution in the range $(-0.20, 0.20)$.

Walls block the robot's motion, i.e., when the robot tries to execute an action that would crash it into a wall, the action keeps the robot in its original position.

The robot knows its location in the space through continuous coordinates (x, y). We assume that we have the optimal uniform discretization of the state space (which consists of 24×21 regions) [1]. The goal in this domain is to reach the area marked with 'G', in a maximum of H actions. When the robot reaches it, it is considered a successful episode, and it receives a reward of 1. Otherwise, it receives a reward of 0.

[1]Different methods for function approximation have been successfully applied on this domain [4]. We have simplified the state space representation to a uniform discretization to focus on the study of Policy Reuse.



(a) Task Ω_1 (b) Task Ω_2 c) Task Ω_3

(d) Task Ω_4 (e) Task Ω_5 (f) Task Ω

Figure 1: Grid-based Office Domain

Figure 1 shows 6 different tasks, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5 and Ω, given that the goal states, and therefore, the reward functions, are different. All these tasks are used in the experiments described in the next sections.

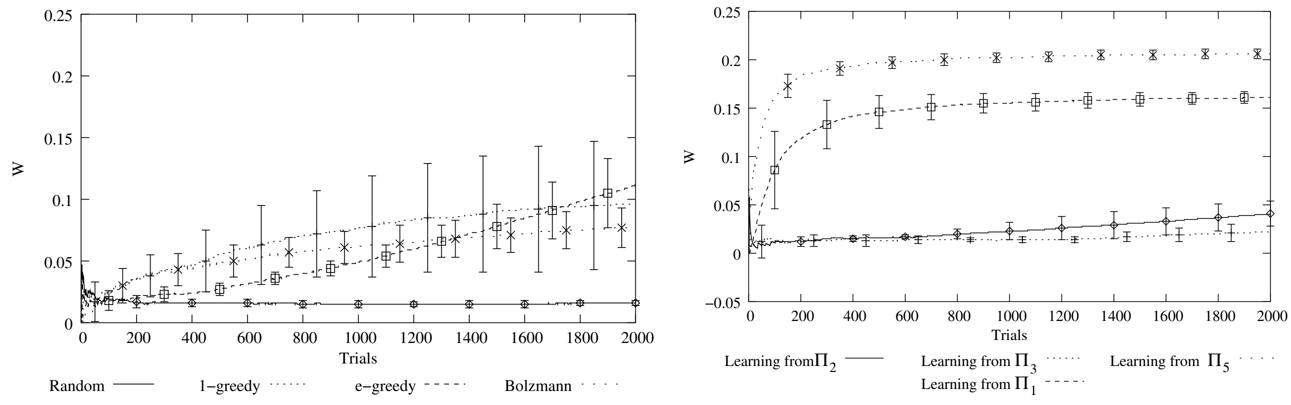
We choose the robot navigation domain for experimentation because it has been widely used in transfer learning papers (e.g., [17, 8, 11]). Transfer learning in more complex domains, as the Keepaway task in robot soccer, requires a mapping between tasks that use different state and action spaces [15, 14]. Such mapping requires a considerable amount of expert knowledge. Our ongoing research line includes the study of how our policy reuse algorithm can be extended to include transfer learning knowledge.

3.2.2 Results

We describe the experimental results of applying different exploration strategies for learning the task Ω, shown in Figure 1(f). Learning has been performed using the Q-Learning algorithm, for fixed parameters of $\gamma = 0.95$ and $\alpha = 0.05$, which empirically have demonstrated to be accurate for learning in this domain.

We analyze the learning performance in executing $K = 2000$ episodes. Each episode consists of following the defined strategy until the goal is achieved or until the maximum number of steps ($H = 100$) is reached. The x axis and the y axis show the episode or trial number and the average gain obtained respectively. A value of 0.2 for the episode 200 means that the average gain obtained in the first 200 episodes is 0.2. The results provided are the average of ten executions. Error bars provide the standard deviation in the ten executions.

The learning process has been first executed following different exploration strategies that do not use any past policy. Specifically, we have used four different strategies: (i) random; (ii) completely greedy; (iii) an ϵ-greedy (i.e., with probability ϵ follows the greedy strategy, and with probability $(1-\epsilon)$ acts randomly), with an initial value of $\epsilon = 0$, which is incremented by 0.0005 in each episode; (iv) the



(a) Learning from Scratch (b) Learning with π-Reuse

Figure 2: Results of the learning process for different exploration strategies.

Boltzmann strategy ($P(a_j) = \frac{e^{\tau Q(s,a_j)}}{\sum_{p=1}^{n} e^{\tau Q(s,a_p)}}$), initializing $\tau = 0$, and increasing it by 5 in each learning episode.

Figure 2(a) shows the results. When acting randomly, the average gain in learning is almost 0, given that acting randomly is a very poor strategy. However when a greedy behavior is introduced, (strategy 1-greedy), the curve shows a slow increment, achieving values of almost 0.1. The curve obtained by the Boltzmann strategy does not offer significant improvements. The ϵ-greedy strategy seems to compute an accurate policy in the initial episodes, and it corresponds to the highest average gain at the end of the learning.

We then execute the learning with policy reuse. Figure 2(b) shows the learning curves of different learning processes performed with the π-reuse exploration strategy. In each of them, a different policy has been reused. In the first one, the policy reused is Π_5 ($\Pi_{past} = \Pi_5$), which is optimal to solve the task Ω_5. Such task has its goal into the same room as the goal of Ω. In the second case, $\Pi_{past} = \Pi_1$, which solved Ω_1. In the last two cases, $\Pi_{past} = \Pi_2$ and Π_3 respectively, whose associated tasks (Ω_2 and Ω_3) are very different when compared with Ω. All the reused policies are optimal for their respective tasks.

The parameters used in the Q-Learning update equation are the same as above ($\gamma = 0.95$ and $\alpha = 0.05$). The parameter setting for the π-reuse exploration strategy are $\psi = 1$, $\upsilon = 0.95$, and $\epsilon = 1 - \psi_h$. [2]

Figure 2(b) shows how, when biasing the exploration process for learning the task Ω with the policies Π_1 and Π_5, the obtained gain significantly increases within the first few episodes of the execution. For instance, when reusing Π_5, in only 100 iterations the average gain is higher than 0.15, and after 400 iterations the value stays around 0.2. When reusing Π_1, the gain is higher than 0.1 after only 200 episodes, and after 500 episodes it stays around 0.15. In both cases, the standard deviation is high in the initial episodes, but it approaches 0 in subsequent episodes.

However when the learning is biased with a very different policy, as Π_2 and Π_3, the average gain shown in Figure 2(a) is below 0.05, so the learning process is even worse than when learning from scratch.

The results show that reusing a past policy provides a bias in the exploration process which can speed up the learning when compared with learning from scratch. The improvement depends on whether the reused policy solves a task similar to the one we are currently learning. A similarity metric between tasks is hard to define even for a well structured domain.

Interestingly, we introduce a similarity metric between policies based on the usefuness of reuse. For instance, the results in Figure 2(b) show that the learning curves provide us information on the similarity between policies. In that figure, the gain obtained for each of the past policies can be understood as: (i) an estimation of how similar the policy reused is to the one we are currently learning; and (ii) an estimation of how useful the policy reused is in order to learn the new policy. Actually, the gain obtained by each past policy can be used to rank the similarity of the past policies with respect to the new one. In this case, the most similar policy to Π_Ω is Π_5, followed by Π_1, Π_2 and Π_3.

Furthermore, the gain estimated above can be computed very fast, and in only 25 episodes, the gain of reusing the policy Π_5 significantly outperforms the gain of reusing the other policies.

4. REUSING A LIBRARY OF POLICIES

We describe now the PRQ-learning algorithm for the efficient reuse of the policies stored in a Policy Library. We describe a similarity function that estimates the usefulness of reusing a specific policy for learning a new one.

4.1 A Similarity Function Between Policies

The exploration strategy π-reuse, as defined in Table 1, returns the learned policy Π_{new}, and the average gain obtained in its learning process, W. Let W_i be the gain obtained while executing the π-reuse exploration strategy, reusing the past policy Π_i. We can use such value to measure the usefulness of reusing the policy Π_i to learn the new policy Π_{new}. The next definitions formalize this idea.

Definition 4. *Given a policy Π_i that solves a task $\Omega_i =< \mathcal{D}, R_i >$, and a new task $\Omega =< \mathcal{D}, R_\Omega >$, the Reuse Gain of*

[2]This parameter setting has been chosen after an informal experimentation, so different values of the parameters could provide better results. However they are good enough to evaluate the algorithm and to move on the next steps of the research on Policy Reuse. Additional information about the properties of this parameter setting can be found in [5].

the policy Π_i on the task Ω, W_i, is the gain obtained when applying the π-reuse exploration strategy with the policy Π_i to learn the policy Π.

Then the most useful policy to reuse, Π_k, from a Library Policy, $L = \{\Pi_1, \ldots, \Pi_n\}$, is the one that maximizes the Reuse Gain when learning such a task, as defined in equation 3:

$$\Pi_k = \arg_{\Pi_i} \max(W_i), i = 1, \ldots, n \tag{3}$$

To solve this equation we need to compute the Reuse Gain for all the past policies. Interestingly, such a gain can be estimated on-line at the same time that the new policy is computed. This idea is formalized in the PRQ-Learning algorithm.

4.2 The PRQ-Learning Algorithm

The goal of the PRQ-learning algorithm is to solve a task Ω, i.e. to learn an action policy Π_Ω. We have a Policy Library $L = \{\Pi_1, \ldots, \Pi_n\}$ composed of n past optimal policies that solve n different tasks respectively. Then two main questions need to be answered: (i) given the set of policies $\{\Pi_\Omega, \Pi_1, \ldots, \Pi_n\}$, which consists of the policies in the Policy Library plus the ongoing learned policy, what policy is exploited? (ii) once a policy is selected, what exploration/exploitation strategy is followed?

The answer to the first question is as follows: let W_i be the Reuse Gain of the policy policy Π_i on the task Ω. Also, let W_Ω be the average reward that is received when following the policy Π_Ω greedily. The solution we introduce consists of following a softmax strategy using the values W_Ω and W_i, as defined in equation 4, with a temperature parameter τ. This value is also computed for Π_0, which we assume to be Π_Ω. Equation 4 provides a way to decide whether to exploit the past policies or the new one.

$$P(\Pi_j) = \frac{e^{\tau W_j}}{\sum_{p=0}^{n} e^{\tau W_p}} \tag{4}$$

The answer to the second question (what exploration strategy to follow once a policy is chosen) is an heuristic that depends on the selected policy. If the policy chosen is Π_Ω, the algorithm follows a completely greedy strategy. However, if the policy chosen is Π_i (for $i = 1, \ldots, n$), the π-reuse action selection strategy, defined in previous section, is followed instead. In this way, the Reuse Gain of each of the past policies can be estimated on-line with the learning of the new policy. Thus, the values required in Equation 4 are continuously updated each time a policy is used.

All these ideas are formalized in the PRQ-Learning algorithm (Policy Reuse in Q-Learning) shown in Table 2. The algorithm gets as input: a new task to solve Ω; the policy library L; the temperature parameter of the softmax policy selection equation τ, and a decay parameter $\Delta\tau$; and a set of previously defined parameters: $K, H, \psi, \upsilon, \gamma, \alpha$.

The algorithm initializes the new Q function to 0, as well as the estimated reuse gain of the policies in the library. Then the algorithm executes the K episodes iteratively. In each episode, the algorithm decides which policy to follow. In the first iteration, all the policies have the same probability to be chosen, given that all W_i values are initialized to 0. Once a policy is chosen, the algorithm uses it to solve the task, updating the Reuse Gain for such a policy with the reward obtained in the episode, and therefore, updating the probability to follow each policy. The policy being learned can also be chosen, although in the initial steps it behaves as a random policy, given that the Q values are initialized to 0. While new updates are performed over the Q function, it becomes more accurate, and receives higher rewards when executed. After executing several episodes, it is expected that the new policy obtains higher gains than reusing the past policies, so it will be chosen most of the time.

```
PRQL(Ω, L, τ, Δτ, K, H, ψ, υ, γ, α)

• Given:
    1. A new task $\Omega$ we want to solve
    2. A Policy Library $L = \{\Pi_1, \ldots, \Pi_n\}$
    3. An initial value of the temperature parameter, $\tau$, and an
       incremental size, $\Delta\tau$, for the Boltzmann policy selection
       strategy
    4. A maximum number of episodes to execute, $K$
    5. A maximum number of steps per episode, $H$
    6. The parameters $\psi$ and $\upsilon$ for the $\pi$-exploration strategy
    7. The parameters $\gamma$ and $\alpha$ for the Q-learning update equation
• Initialize:
    1. $Q_\Omega(s,a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
    2. Initialize $W_\Omega$ to 0
    3. Initialize $W_i$ to 0
    4. Initialize the number of episodes where policy $\Pi_\Omega$ has
       been chosen, $U_\Omega = 0$
    5. Initialize the number of episodes where policy $\Pi_i$ has been
       chosen, $U_i = 0, \forall i = 1, \ldots, n$
• For $k = 1$ to $K$ do
    – Choose an action policy, $\Pi_k$, assigning to each policy the
      probability of being selected computed by the following
      equation (equation 4):
          $P(\Pi_j) = \frac{e^{\tau W_j}}{\sum_{p=0}^{n} e^{\tau W_p}}$
      where $W_0$ is set to $W_\Omega$
    – Execute the learning episode $k$
        * If $\Pi_k = \Pi_\Omega$, execute a Q-Learning episode following
          a fully greedy strategy
        * Otherwise, use the $\pi$-reuse exploration strategy to
          reuse $\Pi_k$, i.e. call $\pi$-reuse$(\Pi_k, 1, H, \psi, \upsilon)$
        * In any case, receive the reward obtained in that
          episode, say $R$, and the updated Q function, $Q_\Omega(s,a)$
    – Set $W_k = \frac{W_k U_k + R}{U_k + 1}$
    – Set $U_k = U_k + 1$
    – Set $\tau = \tau + \Delta\tau$
• Return the policy derived from $Q_\Omega(s,a)$
```

Table 2: PRQ-Learning.

4.3 Experiments

We present the experiments performed with the PRQ-Learning algorithm. We demonstrate three main claims: Firstly, that the performance can be improved if we can bias the exploration with past policies, even if we have several and we do not know apriori which one is the most similar and/or useful to reuse. Second, that it is possible to determine which is the most useful policy simultaneously while learning the new policy. And third, that a balance between exploring, exploiting past policies, and exploiting the new policy being learned can be successfully achieved.

We use the PRQ-Learning algorithm for learning the task Ω, defined in Figure 1(f). We assume that we have 3 different libraries of policies, so we distinguish three different cases. In the first one, the policy library is $L_1 = \{\Pi_2, \Pi_3, \Pi_4\}$, assuming that the tasks Ω_2, Ω_3 and Ω_4, defined in Figure 1(b), (c) and (d) respectively, were previously solved. All these tasks are very different from the one we want to solve, so their policies are not supposed to be very useful in learning the new one. In the second case, Π_1 is added, so $L_2 = \{\Pi_1, \Pi_2, \Pi_3, \Pi_4\}$. The third case uses the Policy Library $L_3 = \{\Pi_2, \Pi_3, \Pi_4, \Pi_5\}$

The PRQ-Learning algorithm is executed for the three cases. The learning curves are shown in Figure 3. The parameters used are the same used in Section 3.2. The only new parameters are the ones of the Boltzmann policy selection strategy, $\tau = 0$, and $\Delta\tau = 0.05$, obtained empirically.

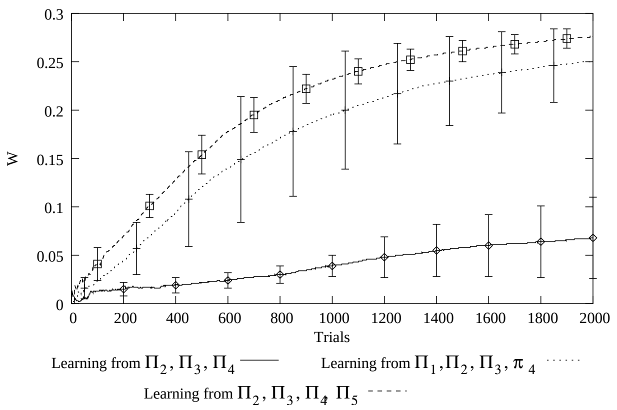


Figure 3: Learning curve when learning the task of Figure 1(f) reusing different sets of policies.

Figure 3 shows two main conclusions. Firstly, when a very similar policy is included in the set of policies to be reused, the improvement on learning is very high. For instance, when reusing Π_1 and Π_5, the average gain is greater than 0.1 in only 500 iterations, and more than 0.25 at the end of the episode. Secondly, when no similar policy is available, the learning curve is similar to the results obtained when learning from scratch with the 1-greedy strategy (which is the strategy followed by PRQ-Learning for the new policy, as defined by the PRQ-Learning algorithm). This demonstrates that the PRQ-learning algorithm has discovered that reusing the past policies is not useful, so it follows the best strategy available, which is to the 1-greedy strategy with the new policy.

In summary, we can say that the PRQ-learning algorithm has demonstrated to successfully reuse a predefined set of policies. The remaining issue consists of investigating how to acquire such a set of policies. The next section focuses on how an agent can build a library of policies.

5. BUILDING A LIBRARY OF POLICIES

We now describe the *PLPR* algorithm (Policy Library through Policy Reuse), an algorithm to build a library of policies. The algorithm is based on an incremental learning of policies that solve different tasks. The tasks to be solved are not known apriori, and are sequentially given. Otherwise, a method to learn them in parallel could be applied [9].

5.1 The PLPR Algorithm

The PLPR algorithm works as follows. Initially the Policy Library is empty, $PL = \emptyset$. When the first task Ω_1 is solved, the corresponding learned policy Π_1 is learned without reuse. Π_1 is added to the Policy Library and $PL = \{\Pi_1\}$. When a second task needs to be solved, the PRQ-Learning algorithm is applied reusing Π_1. Π_2 is learned. The algorithm makes a decision on whether to add Π_2 to the Policy Library or not. This decision is based on how similar Π_1 is to Π_2, following the similarity function we now introduce.

Definition 5. *Given a policy, Π_i that solves a task $\Omega_i =< \mathcal{D}, R_i >$, a new task $\Omega =< \mathcal{D}, R_\Omega >$, and its respective optimal policy, Π, Π is δ-similar to Π_i (for $0 \leq \delta \leq 1$) if $W_i > \delta W^*_\Omega$, where W_i is the Reuse Gain of Π_i on task Ω and W^*_Ω is the average gain obtained in Ω when an optimal policy is followed.*

The interesting aspect of this concept is that for any optimal policy Π, if we know a past policy which is δ-similar to it, we also know that such optimal policy can be easily learned just by applying the π-reuse algorithm with the past policy. The gain obtained in the learning process (the reuse gain) will be at least δ times the maximum gain in such a task. From this definition, we can formalize the concept of δ-similarity with respect to a Policy Library, L.

Definition 6. *Given a Policy Library, $L = \{\Pi_1, \ldots, \Pi_n\}$ in a domain $\mathcal{D}$, a new task $\Omega =< \mathcal{D}, R_\Omega >$, and its respective optimal policy, Π, Π is δ-similar with respect to L iff $\exists \Pi_i$ such as Π is δ-similar to Π_i, for $i = 1, \ldots, n$.*

If the algorithm finds that a policy Π is δ-similar with respect to a Policy Library L, then it knows that the policy Π can be easily learned by reusing the policies in L. Table 3 presents the PLPR algorithm, which is executed each time that a new task needs to be solved. It gets as an input the Policy Library and the new task to solve, and outputs the learned policy and the updated Policy Library.

PLPR Algorithm

- Given:
  1. A Policy Library, L, composed of n policies, $\{\Pi_1, \ldots, \Pi_n\}$
  2. A new task Ω we want to solve
  3. A δ parameter
- Execute the PRQ-Learning algorithm reusing L. Receive from this execution Π_Ω, W_Ω and W_{max}, where:
  - Π_Ω is the learned policy
  - W_Ω is the average gain obtained when the policy Π_Ω was followed
  - $W_{max} = \max W_i$, for $i = 1, \ldots, n$
- Update PL using the following equation:

$$L = \begin{cases} L \cup \{\Pi_\Omega\} & \text{if } W_{max} < \delta W_\Omega \\ L & \textit{otherwise} \end{cases} \quad (5)$$

Table 3: PLPR Algorithm.

Equation 5 is the update equation for the Policy Library. It requires the computation of the most similar policy in the library, which is the policy Π_j such as $j = \arg_i \max W_i$, for $i = 1, \ldots, n$. W_{max} is the gain obtained by reusing such a policy. The new policy learned is inserted in the library if W_{max} is lower than δ times the gain obtained by using the new policy (W_Ω), where $\delta \in [0, 1]$ defines the similarity threshold, i.e., whether the new policy is δ-similar with respect to the Policy Library.

The parameter δ hence plays an important role. If $\delta = 0$, the Policy Library stores only the first policy learned, given that the average gain obtained by reusing it will be greater than zero in most cases, due to the positive rewards obtained by chance. If $\delta = 1$, most of the policies learned are inserted, as $W_{max} < W_\Omega$, given that W_Ω is maximum if the optimal policy has been learned. Different δ values in the range $(0, 1)$ provide different sizes of the library. Thus δ defines the "resolution" of the library.

The PLPR algorithm has an interesting side effect in terms of learning the structure of the domain. The Policy Library is initialized to empty, and a new policy is included only if it is different *enough* (depending on the δ threshold) from the previously stored ones. When the policies stored are *fully representative* of the domain, no more policies are stored. Therefore the obtained library can be considered as the *Basis-Library* of the domain, and the stored policies can be considered as the *core-policies* of such domain.

5.2 Experiments

We present the experiments performed to learn a Basis-Library in the navigation domain (Section 3). We perform a task consisting of the executing $K = 2000$ episodes. Like in the previous experiments, each episode consists of a sequence of actions until the goal is achieved or until the maximum number of actions, $H = 100$, is reached.

A number, 50, of different tasks are sequentially performed, each of them with a different goal area, located in different positions of the different rooms of the domain, as shown in Figure 4(a). The figure does not represent a unique task with 50 different goals, but the 50 different goal areas of the 50 different tasks. The results provided are the average of 10 different executions, in which the 50 different tasks are sequentially performed following a random order. In these experiments, we use the same parameter settings as earlier (see Section 4.3).

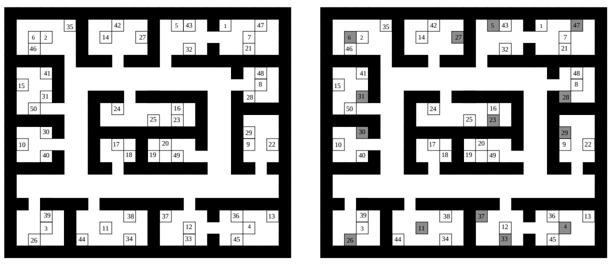

(a) 50 different goal areas (b) Core-Policies

Figure 4: Office Domain.

The first element to study is the size of the Policy Library built while performing the tasks with the PLPR algorithm, i.e., the number of core-policies stored in the Policy Library. Figure 5 shows in the y axis the size of the Policy Library, and in the x axis, the number of tasks performed up to that moment. As introduced above, when $\delta = 0$, only 1 policy is stored. When $\delta = 0.25$, the number of core-policies is around 14. Interestingly, this is very close to the number of rooms in the domain (15). While increasing δ, the number of core-policies increases and when $\delta = 1$, almost all the learned policies are stored.

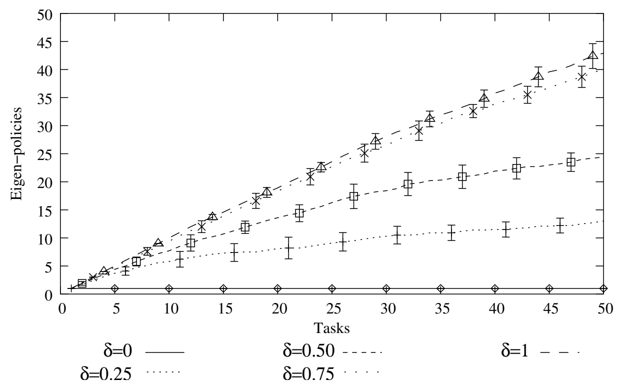


Figure 5: Number of eigen-policies obtained.

Figure 4(b) shows an example of the core-policies obtained in one of the executions with $\delta = 0.25$. The figure represents the resulting Policy Library composed of 14 core-policies. In the figure, we assume that a policy is represented by the goal area of the task that it solves. A core-policy is represented also by a shaded goal area. The figure demonstrates that one and only one core-policy has been learned for most of the rooms. The algorithm has discovered that if two different tasks are given two goal areas in the same room, their respective policies are very similar, so only one of them needs to be stored in the Policy Library. We therefore observe that the PLPR algorithm learns the *structure* of the domain as represented by the core-policies.

Figure 6 shows the average gain obtained when performing the 50 different tasks with the PLPR algorithm, for the different values of δ. In most of the cases, $\delta = 0.25, 0.50, 0.75$ and 1, the average gain increases up to more than 0.2, and no significant differences exist between them. Only in the case of $\delta = 0$, the average gain stays low, around 0.16, given that $\delta = 0$ generates a Policy Library with only one policy (the first one learned).

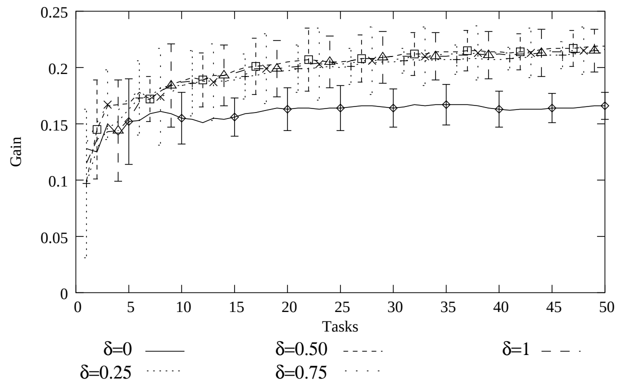


Figure 6: Results of PLPR.

For comparison purposes, we executed the same learning process with different exploration strategies as learning from scratch. In those cases, while new policies are learned from scratch, the average gain obtained stabilizes around 0.12 for all the strategies, without very significant differences. Therefore, Policy Reuse can almost reach a 100% gain in the performance of the 50 tasks over the results obtained when the 50 tasks are learned from scratch.

6. CONCLUSIONS

We have introduced different algorithms to address the main challenges of policy reuse in a reinforcement learning agent. First, the PRQ-Learning algorithm allows to probabilistically bias an exploration learning process by reusing a Policy Library; the algorithm significantly improves the learning performance over exploration strategies that learn from scratch. Second, the PLPR algorithm incrementally builds the Policy Library; the library is built at the same time as new policies are learned and past policies are reused. And last, our method to build the Policy Library allows the learning of the structure of the domain in terms of a set of core-policies.

Future ongoing research includes the extension of Policy Reuse to reuse across different representational frameworks, including different agents or domains. We envision multiple agents sharing and reusing policies previously learned to improve the solvability horizon of individual non-reuse based learning agents.

7. ACKNOWLEDGMENTS

This research was conducted while the first author was visiting Carnegie Mellon from the Universidad Carlos III de Madrid, supported by a generous grant from the Spanish Ministry of Education and Fullbright. He was partially sponsored also by the Ministry of Education and Science project number TIN2005-08945-C06-05 and by CAM-UC3M project number UC3M-INF-05-016. The second author was partially sponsored by Rockwell Scientific Co., LLC under subcontract no. B4U528968 and prime contract no. W911W6-04-C-0058 with the US Army, and by BBNT Solutions, LLC under contract no. FA8760-04-C-0002 with the US Air Force. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the sponsoring institutions, the U.S. Government or any other entity.

8. REFERENCES

[1] M. Bowling and M. Veloso. Bounding the suboptimality of reusing subproblems. In *Proceedings of IJCAI-99*, 1999.

[2] J. Carroll and T. Peterson. Fixed vs. dynamic sub-transfer in reinforcement learning. In *Proceedings of the International Conference on Machine Learning and Applications*, 2002.

[3] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[4] F. Fernández and D. Borrajo. On determinism handling while learning reduced state space representations. In *Proceedings of the European Conference on Artificial Intelligence (ECAI 2002)*, Lyon (France), July 2002.

[5] F. Fernández and M. Veloso. Exploration and policy reuse. Technical Report CMU-CS-05-172, School of Computer Science, Carnegie Mellon University, 2005.

[6] B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[7] R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005. To appear.

[8] M. G. Madden and T. Howley. Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review*, 21:375–398, 2004.

[9] R. B. Ollington and P. W. Vamplew. Concurrent Q-Learning: Reinforcement learning for dynamic goals and environments. *International Journal of Intelligent Systems*, 20:1037–1052, 2005.

[10] B. Price and C. Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

[11] A. A. Sherstov and P. Stone. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005.

[12] Ö. Şimşek, A. P. Wolfe, and A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, 2005.

[13] R. S. Sutton, D. Precup, and S. Singh. Intra-option learning about temporally abstract actions. In *Proceedings of the Internacional Conference on Machine Learning (ICML'98)*, 1998.

[14] M. E. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, July 2005. To appear.

[15] M. E. Taylor, P. Stone, and Y. Liu. Value functions for RL-based behavior transfer: A comparative study. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005. To appear.

[16] S. Thrun. Efficient exploration in reinforcement learning. Technical Report C,I-CS-92-102, Carnegie Mellon University, January 1992.

[17] S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems 7*. MIT Press., 1995.

[18] W. T. B. Uther. *Tree Based Hierarchical Reinforcement Learning*. PhD thesis, Carnegie Mellon University, August 2002.

[19] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.