Probabilistic Reuse of Past Policies

Fernando Fernández **Manuela Veloso**

July 2005
CMU-CS-05-173

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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. 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.

Keywords: Reinforcement Learning, Policy Reuse, Transfer Learning.

Abstract

A past policy provides a bias to guide the exploration of the environment and speed up the learning of a new action policy. The success of this bias depends on whether the past policy is "similar" to the actual policy or not. In this report we describe a new algorithm, PRQ-Learning, that reuses a set of past policies to bias the learning of a new one. The past policies are ranked following a similarity metric that estimates how useful is to reuse each of those past policies. This ranking provides a probabilistic bias for the exploration in the new learning process. Several experiments demonstrate that PRQ-Learning finds a balance between exploitation of the ongoing learned policy, exploration of random actions, and exploration toward the past policies.

1 Introduction

Reinforcement Learning [7] is a widely used tool to learn to solve different tasks in different domains. By *domain* we mean the rules that define how the actions of the learning agent influence the environment, i.e. the state transition function. By *task* we mean the specific problem that the agent is trying to solve in the domain, which is defined through the reward function.

The goal of this work is to study how action policies that are learned to solve a defined set of tasks can be used to solve a new and previously unseen task. A first approach is to use a policy through the transfer of the Q function. The past Q function is used to seed the learning of the new one. However, any past policy when followed greedily, provides a whole plan that maximizes the expected reward in the past task. This plan depends on the domain and the past task but not on the new task. Thus, Q function transfer between tasks is useful when the reward functions of the new and old tasks are very similar, but provides very poor results if they are different [2].

There are other areas in RL in which sub-policies are reused. For instance, some algorithms use macro-actions to learn new action policies in Semi-Markov Decision Processes, as it is the case of TTree [11] and Intra-Option Learning [8]. Hierarchical RL uses different abstraction levels to organize subtasks [3].

Policy Reuse [5] is a learning technique guided by past policies to balance among exploitation of the ongoing learned policy, exploration of random actions, and exploration toward the past policies. Thus, it is very related with the exploration vs. exploitation problem, which tries to define whether to explore new or exploit the knowledge already acquired. In the literature, different kinds of exploration strategies can be found. A random strategy always selects randomly the action to execute. The ϵ-greedy strategy selects with a probability of ϵ the best action suggested by the Q function learned up to that moment, and it selects a random action with probability of $(1 - \epsilon)$. In alternative, Boltzmann strategy ranks the actions to be used, providing with a higher probability to the actions with a higher value of Q. Directed exploration strategies memorize exploration-specific knowledge that is used for guiding the exploration search [9]. These strategies are based in heuristics that bias the learning so unexplored states tend to have a higher probability of being explored that recently visited ones. None of these strategies include knowledge of past policies, but knowledge obtained in the current learning process.

Nevertheless, several examples found in the AI bibliography have demonstrated that information of past problems can be useful for solving new ones, as Policy Reuse does. For instance, past plans can be used to guide the search of new ones through control rules in a planning system [12]. Also, way-points followed in past paths can be used to bias the search of new paths in a path-planning system, and to speed up the search [1].

In this work, we contribute PRQ-Learning, an algorithm that implements Policy Reuse ideas efficiently. This algorithm allows us to reuse past policies to learn a new one, improving the results of learning from scratch. The improvement is achieved without prior knowledge about which policies are useful, and not even knowing whether a useful one exists or not.

The report is organized as follows. Section 2 describes the main elements of Policy Reuse. Firstly, the concepts of domain and task are related with Markov Decision Processes. Second, Policy Reuse is formally defined. Third, the π-reuse exploration strategy is introduced, which is able to balance the exploration of new actions, the exploitation of the current policy, and the exploitation of a past predefined policy [5]. And last, the concept of similarity between policies is

motivated.

Section 3 introduces the PRQ-Learning algorithm. The experiments described in Section 4 demonstrate three capabilities of the PRQ-Learning algorithm. Firstly, that a ranking of similarity between past policies can be estimated simultaneously to learning the new policy. Second, that PRQ-Learning is able to use the previously defined ranking to find a correct balance among exploiting past policies, exploring new actions, or exploiting the policy that is currently being learned. And third, that PRQ-Learning can improve learning performance when compared with learning from scratch. Lastly, Section 5 concludes with new research lines.

2 Domains, Tasks and Policy Reuse

The goal of this section is to introduce Policy Reuse. To do this, we first describe the concepts of task, domain, and gain. Then, we define how the reuse of a past policy is used as a bias in a new exploratory process. Last, we define a similarity concept between policies, which motivation is deeply described in [5].

2.1 Domain, Tasks and MDPs

A Markov Decision Process [6] is represented with a tuple $< \mathcal{S}, \mathcal{A}, \delta, \mathcal{R} >$, where $\mathcal{S}$ is the set of all possible states, $\mathcal{A}$ is the set of all possible actions, δ is an unknown stochastic state transition function, $\delta : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re$, and $\mathcal{R}$ is an unknown stochastic reward function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$. We focus in RL domains where different *tasks* can be solved. We introduce a task as a specific reward function, but the other concepts, $\mathcal{S}$, $\mathcal{A}$ and δ stay constant for all the tasks. Thus, we extend the concept of an MDP introducing two new concepts: domain and task. We characterize a domain, $\mathcal{D}$, as a tuple $< \mathcal{S}, \mathcal{A}, \delta >$. We define a task, Ω, as a tuple $< \mathcal{D}, \mathcal{R}_\Omega >$, where $\mathcal{D}$ is a domain as defined before, and $\mathcal{R}_\Omega$ is the stochastic and unknown reward function.

In this work we assume that we are solving a task with absorbing goal states. Thus, if s_i is a goal state, $\delta(s_i, a, s_i) = 1$, $\delta(s_i, a, s_j) = 0$ for $s_i \neq s_j$, and $\mathcal{R}(s_i, a) = 0$, for all $a \in \mathcal{A}$. A trial starts by locating the learning agent in a random position in the environment. Each trial finishes when a goal state is reached or when a maximum number of steps, say H, is achieved. Thus, the goal is to maximize the expected average reinforcement per trial, say 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 trial k, in a total of K trials.

An action policy, $\Pi : \mathcal{S} \rightarrow \mathcal{A}$, defines for each state, the action to execute. The action policy Π^* is optimal if it maximizes the gain W in such a task, say W^*_Ω. Action policies can be represented using the action-value function, $Q^\Pi(s, a)$ that 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 translated to learning the previous function, $Q^\Pi(s, a)$. This learning can be performed using different algorithms, as Q-Learning [13].

2.2 Policy Reuse

The goal of Policy Reuse is to describe how learning can be sped up if different policies, which solve different tasks, are used to bias the exploration process of the learning of the action policy of another similar task. Then, the scope of this work is summarized as:

- We need to solve the task Ω, i.e. learn Π^*_Ω.
- We have previously solved the set of tasks $\{\Omega_1, \ldots, \Omega_n\}$, so we have the set of policies, $\{\Pi^*_1, \ldots, \Pi^*_n\}$, to solve them respectively.
- How can we use the previous policies, Π^*_i to learn the new one, Π^*_Ω?

To solve this problem we have developed the PRQ-Learning algorithm. This algorithm automatically answer two questions: (i) what policy, from the set $\{\Pi^*_1, \ldots, \Pi^*_n\}$, is used to bias the new learning process? (ii) once a policy Π_s is selected, how is it integrated in the learning process? The algorithm is based on an exploration strategy, π-reuse, which is able to bias the learning of a new policy with only one past policy. From this strategy, a similarity metric between policies is obtained, providing a method to select the most accurate policy to reuse. Both the π-reuse strategy and the similarity metric, defined in [5], are summarized in the next subsections.

2.3 Exploiting a Past Policy

Reusing a defined past policy requires integrating the knowledge of the past policy into the current learning process. Our approach is to bias the exploratory process of the new policy with the past one.

We denote the old policy with Π^{old}, and the one we are currently learning with Π. We assume that we are using a direct RL method to learn the action policy, so we are learning its related Q function. Any 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 [13].

The goal of the π-reuse strategy is to balance random exploration, exploitation of the past policy, and exploitation of the new policy, which is being learned currently. The π-reuse strategy follows the past policy with a probability of ψ. However, with a probability of $1-\psi$, it exploits the new policy. Obviously, random exploration is always required, so when exploiting the new policy, it follows an ϵ-greedy strategy, as is defined in Table 1. Lastly, the υ parameter allows the decay of the value of ψ in each trial.

The interesting of the π-reuse estrategy is that it also contributes a similarity metric among policies, as it is summarized in the next subsection.

2.4 A Similarity Metric 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. Let's call W_i to the gain obtained while executing the π-reuse exploration strategy, reusing the past policy Π_i.

We call Π^*_Ω the optimal action policy for solving the task Ω. W^*_Ω is the gain obtained when using the optimal policy, Π^*_Ω, to solve Ω. Therefore, W^*_Ω is the maximum gain that can be obtained

π-reuse ($\Pi_{old}, K, H, \psi, \upsilon$).
for $k = 1$ to K
Set the initial state, s, randomly.
Set $\psi_1 \leftarrow \psi$
for $h = 1$ to H
With a probability of ψ_h, $a = \Pi_{old}(s)$
With a probability of $1 - \psi_h$, $a = \epsilon$-greedy$(\Pi_{new}(s))$
Receive current state s', and reward, $r_{k,h}$
Update $Q^{\Pi_{new}}(s, a)$, and therefore, Π_{new}
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 and Π_{new}

Table 1: π-reuse Exploration Strategy.

in Ω. Then, we can use the difference between W^*_Ω and W_i to measure how useful to reuse the policy Π_i is to learn to solve the new task, using the distance metric shown in equation 2.

$$d_{\rightarrow}(\Pi_i, \Pi) = W^*_\Omega - W_i \tag{2}$$

Then, the most useful policy to reuse, from a set $\{\Pi_1, \ldots, \Pi_n\}$, is:

$$\arg_{\Pi_i} \min(W^*_\Omega - W_i), i = 1, \ldots, n \tag{3}$$

However, W^* is independent of i, so the previous equation is equivalent to:

$$\arg_{\Pi_i} \max(W_i), i = 1, \ldots, n \tag{4}$$

This equation is not possible to compute, given that the set of W_i values, for $i = 1, \ldots, n$ is unknown a priori. However, it can be estimated on-line at the same time that the new policy is computed. This idea is formalized in the PRQ-Learning algorithm.

3 PRQ-Learning Algorithm

We are focused on learning to solve a task Ω, i.e. to learn an action policy Π_Ω. We have n past optimal policies to solve n different tasks respectively. For simplicity in the notation, we will call these policies $\Pi_1, \ldots, \Pi_n$, and $\Omega_1, \ldots, \Omega_n$ the tasks. Also, let's call $W_i^{x_i}$ the expected average reward that is received when following the policy Π_i and using an action selection strategy x_i. This strategy could be Boltzmann, π-reuse or any other strategy. Also, let's call W^x_Ω the average reward that is received when following the policy Π_Ω and using an action selection strategy x.

When deciding which action to execute in each step of the learning process of the policy Π_Ω, the following decisions must be taken: (i) what policy is followed from the set $\{\Pi_\Omega, \Pi_1, \ldots, \Pi_n\}$? (ii) once a policy is selected, what exploration/exploitation strategy is followed?

The answer proposed to the first question is to follow a softmax strategy, using the values W_Ω^x and $W_i^{x_i}$, as defined in equation 5, where a temperature parameter τ is included. Notice also that this value is also computed for Π_0, which we assume to be Π_Ω.

$$P(\Pi_j) = \frac{e^{\tau W_j^{x_i}}}{\sum_{p=0}^{n} e^{\tau W_p^{x_p}}} \tag{5}$$

Once the policy to follow has been chosen, whether to follow it greedily, or to introduce also an exploratory element, must be decided, i.e. we need to decide x and x_i, for $i = 1, \ldots, n$. If the policy chosen is Π_Ω, a completely greedy strategy is followed. However, if the policy chosen is Π_i ($i = 1, \ldots, n$), the π-reuse action selection strategy, defined in previous section, is followed. The whole algorithm, which we have called PRQ-Learning (Policy Reuse in Q-Learning) is shown in Table 2. The learning algorithm used is Q-Learning. It has been chosen because it is an off-policy algorithm. Any other off-policy algorithm could be chosen.

4 Experiments

In this section we demonstrate three main results. First, given a set of past policies, the most similar policy to the new one can be learned simultaneously to learning the new policy. Second, a balance between exploring new actions, exploiting past policies, and exploiting the new policy that is being learned currently is successfully achieved. And third, performance can be improved if we can bias the exploration with past policies even if: (a) we have several past policies, (b) we do not know a priori which one is the most similar. The next subsection describes the application domain.

4.1 Navigation Domain

This domain consists of a robot moving inside of an office area, as shown in Figure 1, similar to the one used in other RL works [4, 10]. 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 this case). The possible actions that the robot can execute are "North", "East", "South" and "West", all of size one. The final position after each action is noised by a random variable following a uniform distribution in the range $(-0.20, 0.20)$. The robot knows its location in the space through continuous coordinates (x, y) provided by some localization system. In this work, we assume that we have the optimal uniform discretization of the state space (which consists of 24×21 regions). Furthermore, the robot has an obstacle avoidance system that blocks the execution of actions that would crash it into a wall. The goal in this domain is to reach the area marked with 'G'. When the robot reaches it, it is considered a successful trial, and it receives a reward of 1. Otherwise, it receives a reward of 0.

Figure 1 shows six different tasks in the same domain, 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 different tasks will be used in the experiments.

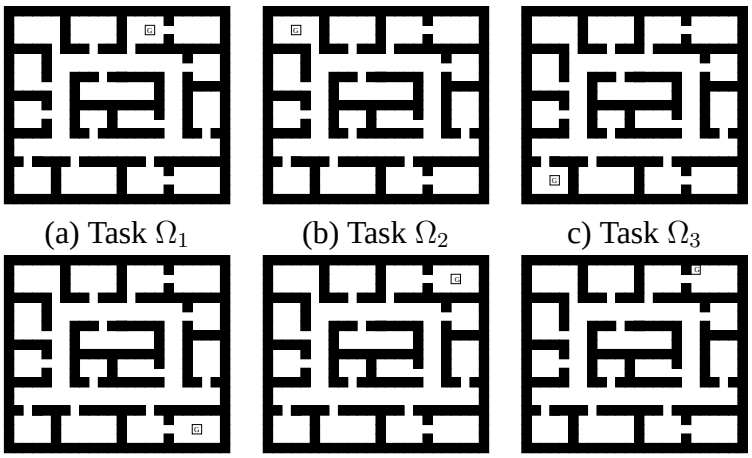
- Given:
  1. A set of n tasks $\{\Omega_1, \ldots, \Omega_n\}$.
  2. Their respective optimal policies, $\{\Pi_1^*, \ldots, \Pi_n^*\}$ to solve them
  3. A new task Ω we want to solve
  4. A maximum number of trials to execute, K
  5. A maximum number of steps per trial, H
- Initialize:
  1. $Q_\Omega(s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
  2. Initialize W_Ω^x to 0
  3. Initialize $W_i^{x_i}$ to 0
  4. Initialize the number of trials where policy Π_Ω has been chosen, $U_\Omega = 0$
  5. Initialize the number of trials where policy Π_i has been chosen, $U_i = 0, \forall i = 1, \ldots, n$
- For $k = 1$ to K do
  - Choose an action policy, Π_j, randomly, assigning to each policy the probability of being selected computed by the following equation (equation 5):
    $$P(\Pi_j) = \frac{e^{\tau W_j^{x_j}}}{\sum_{p=0}^{n} e^{\tau W_p^{x_p}}}$$
  - Initialize the state s to a random state
  - Set $R = 0$
  - for $h = 1$ to H do
    - Use Π_j to compute the next action to execute, a, following the exploitation strategy x_j.
    - Execute a
    - Receive current state, s'
    - Receive current reward, r
    - Update $Q_\Omega(s, a)$ using Q-Learning update function:
      $$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$
    - Set $R = R + \gamma^h r$
    - Set $s \leftarrow s'$
  - Set $W_j^{x_j} = \frac{W_j^{x_j} U_j + R}{U_j + 1}$
  - Set $U_j = U_j + 1$
  - Set $\tau = \tau + \Delta\tau$

Table 2: PRQ-Learning

4.2 Learning Curves

In the following subsections, we will describe different learning processes of a new policy. For each of them we will present two results showing two different curves, the learning curve, and the test curve.

The learning curve of each strategy describes the performance of such strategy in the learning process. Learning has been performed using the Q-Learning algorithm, for fixed parameters of



(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: Office Domain.

$\gamma = 0.95$ and $\alpha = 0.05$, which have been empirically demonstrated to be accurate for learning. A learning trial consists of executing $K = 2000$ trials. Each trial consists of following the defined strategy until the goal is achieved or until the maximum number of steps, $H = 100$, is executed. In the figures containing the curves, the x axis shows the trial number. The y axis represents the gain obtained. Thus, a value of 0.2 for the trial 200 means that the average gain obtained in the 200 first trials has been 0.2.

The test curve represents the evolution of the performance of the policy while it is being learned. Each 100 trials of the learning process, the Q function learned up to that moment is stored. Thus, after the learning process, we can test all those policies. Each test consists on 1000 trials where the robot follows a completely greedy strategy. Thus, the x axis shows the learning trial in which that policy was generated, and the y axis show the result of the test, measured as the average number of steps executed to achieve the goal in the 1000 test trials.

For both the learning and test curves, the results provided are the average of ten executions. In the curves, error bars provide the standard deviation in the ten executions.

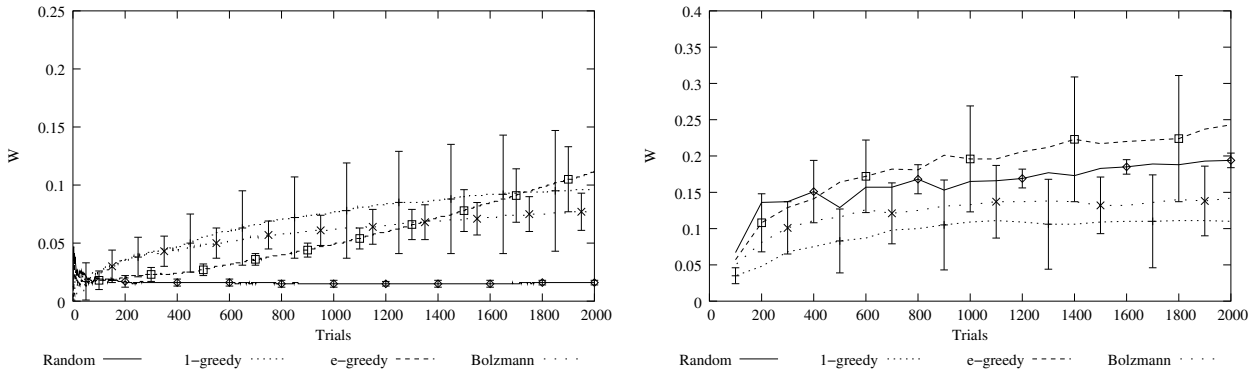
4.3 Learning from Scratch

We want to learn the task described in Figure 1(f). For comparison reasons, the learning and test processes have been executed firstly following different exploratory strategies that do not use any past policy. Specifically, we have used four different strategies. The first one is a random strategy. The second one is a completely greedy strategy. The third one is ϵ-greedy, for an initial value of $\epsilon = 0$, which is incremented by 0.0005 in each trial. Lastly, Boltzmann strategy has been used, initializing $\tau = 0$, and increasing it in 5 in each learning trial. Figure 2 shows the learning and test curves for all of them.

Figure 2(a) shows the learning curve. We see that 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 problem with the 1-greedy strategy is that it also produces a very high standard deviation in the 10 executions performed, showing that a completely greedy strategy may produce very different results. The curve obtained by the Boltzmann strategy do not offer any improvements over ϵ-greedy. However, the ϵ-greedy strategy seems to compute an accurate policy in the initials trials, and obtain the highest average gain at the end of the learning.

The random strategy and ϵ-greedy outperforms the other strategies in the test curve shown in Figure 2(b). This is due to the fact that both strategies, with the defined parameters, are less greedy than the other policies in the initial steps. Typically, higher exploration at the beginning results in more accurate policies.



(a) Learning Curve (b) Test Curve

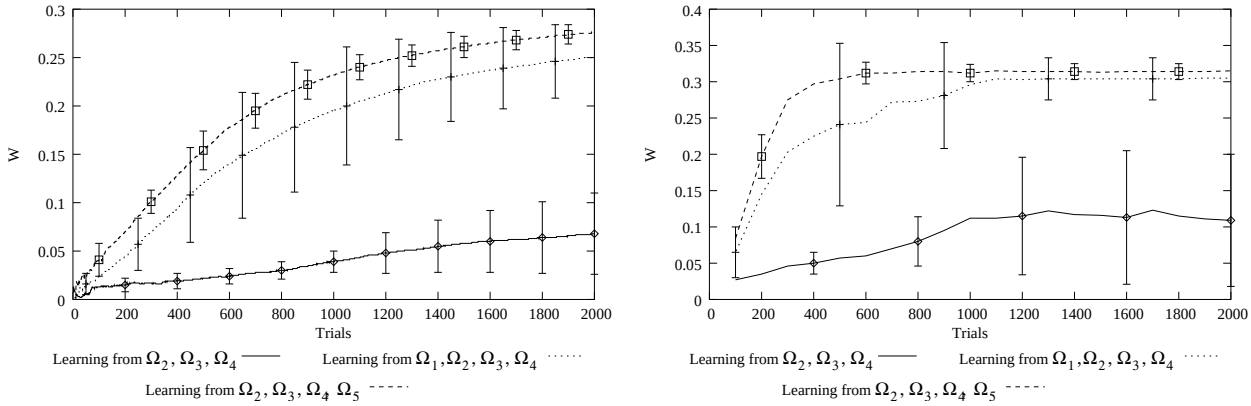
Figure 2: Learning and test evolution when learning from scratch

4.4 Learning with PRQ-Learning

In this section, we introduce the experiments performed with the PRQ-Learning algorithm. In the following we will demonstrate three main issues. Firstly, that performance can be improved if we can bias the exploration with past policies, even if we have several and we do not know, "a priori", which one is the most similar. Second, that which is the most useful policy can be learned simultaneously to learning the new policy. And third, that a balance between exploring, exploiting past policies, and exploiting the new policy that is being learning currently 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 previously learned 3 different set of tasks, so we distinguish three different cases. In the first one, called "Case 1", the past tasks are Ω_2, Ω_3 and Ω_4, defined in Figure 1(b), (c) and (d) respectively. Then, we can use their respective policies, Π_2, Π_3 and Π_4 to bias the learning of the new one, Π_Ω. 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, the set of past policies is also composed with Π_2, Π_3, Π_4, but in this case, the policy Π_1 is also added. The third case uses the policies Π_2, Π_3, Π_4 and Π_5

The PRQ-Learning algorithm is executed for the three cases. The learning curves are shown in Figure 3(a). The parameters used are the same used in Section 2.3. The only new parameters are the ones of the Boltzmann policy selection strategy, $\tau = 0$, and $\Delta\tau = 0.05$, obtained empiri-



(a)Learning Curve (b) Test Curve

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

cally. The result obtained when learning from scratch using Boltzmann exploration strategy is also included for comparison.

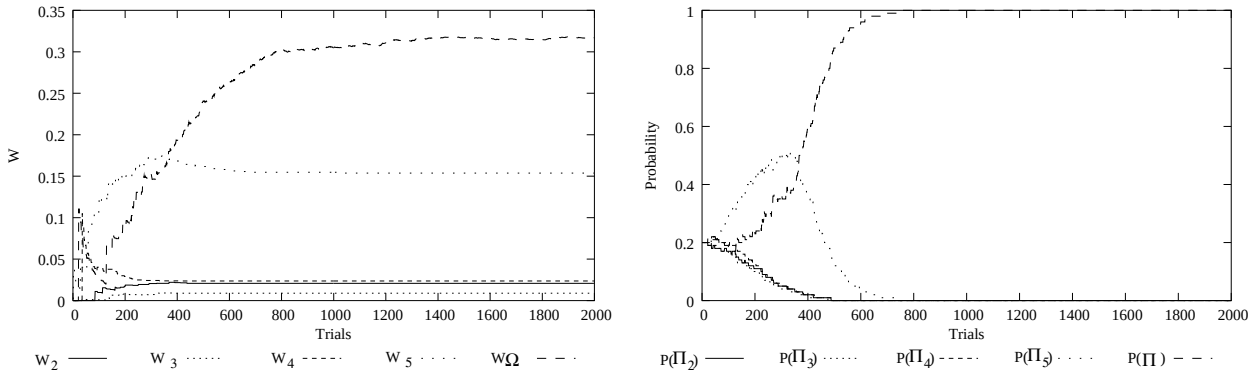
Figure 3(a) shows two main conclusions. On the one hand, when a really similar policy is included in the set of policies that are reused, the improvement on learning is very high. In both cases (when reusing Π_1 and Π_5), average gain is greater than 0.1 in only 500 iterations, and more than 0.25 at the end of the trial. On the other hand, the learning curve when no similar policy is used (case 1) 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 in section 3). That demonstrates that the PRQ-learning algorithm has discovered that reusing the past policies is not useful. Therefore, it follows the best strategy available, which is to follow the 1-greedy strategy with the new policy.

Figure 3(b) shows the test curves for all the cases. The figure shows that when reusing similar past policies in learning, a policy which provides a gain upper than 0.3 is obtained in 1000 trials. That is a strong improvement over the strategies that learn from scratch.

The good results obtained when reusing past similar policies can be easily understood if we look in the Figure 4(a). The figure shows the evolution of the average gain computed for each policy involved, W_5, W_2, W_3, W_4, and W_Ω. That values correspond with one of the learning processes performed when reusing Π_5, Π_2, Π_3, Π_4. It demonstrates how the most similar policy is computed. On the x axis, the number of trials is shown, while the y axis shows the W value for each policy. The figure shows that for Π_2, Π_3 and Π_4, the W values stabilize below 0.05. However, for the policy Π_5, the value increases up to 0.15. The gain of the new policy starts to increase around iteration 100, achieving a value higher than 0.3 by iteration 500.

The gain values computed for each policy are used to compute the probability of selecting them in each iteration of the learning process, using the formula introduced in equation 5, and the parameters introduced above (initial $\tau = 0$, and $\Delta\tau = 0.05$). Figure 4(b) shows the evolution of these probabilities. In the initial steps, all the past policies have the same probability of being chosen (0.2) given that the gain of all them is initialized to 0. While the gain values are updated, only the policy Π_0 stays in a high value, while for the other policies, this value decreases down to 0. However, for the new policy, the value also increases until it achieves the value of 1, given

that its value is the higher after 400 iterations, as shown in Figure 4(a). This demonstrates how the balance between exploiting the past policies or the new one is achieved.



(a)Evolution of W_i (b) Evolution of $P(\Pi_i)$

Figure 4: Evolution of W_i and $P(\Pi_j)$

5 Conclusions and Future Work

In all the works cited in the related work in Section 1, options, macro-actions, and/or policies are used as part of a hierarchy, so learning the new learning process that is performed stays in a higher abstract level above the sub-policies used. The main difference with our work is that we use past policies which are useful by themselves to solve different tasks, and that can help to bias the learning of "similar" ones.

This work contributes an efficient algorithm for policy reuse, PRQ-learning. The algorithm demonstrates that if a useful policy is in the pool of policies available, the algorithm finds it and reuse it efficiently. If no policy is useful, the algorithm also discovers it, and move its behavior to learning from scratch. Thus, the algorithm obtains a correct balance among exploring new actions, exploiting past policies or exploiting the new one. Last, this work opens a wide range of research lines, as policy transfer among different tasks, domains, and/or agents.

References

[1] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002*, Switzerland, October 2002. An earlier version of this paper appears in the Proceedings of the RoboCup-2002 Symposium.

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

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

[4] Fernando Fernández and Daniel 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] Fernando Fernández and Manuela Veloso. Exploration and policy reuse. Technical Report CMU-CS-05-172, School of Computer Science, Carnegie Mellon University, 2005.

[6] M. L. Puterman. *Markov Decision Processes - Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY., 1994.

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

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

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

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

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

[12] Manuela M. Veloso and Jaime G. Carbonell. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10(3):249–278, March 1993.

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