Probabilistic Policy Reuse

Fernando Fernández
Manuela Veloso
Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213. USA

Abstract

We contribute Policy Reuse as a technique to improve a reinforcement learner with guidance from past learned similar policies. Our method relies on using the past policies in a novel way as a probabilistic bias where the learner 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 metric 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. Policy Reuse further contributes the learning of the structure of a domain. Interestingly and almost as a side effect, Policy Reuse identifies classes of similar policies revealing a basis of eigen-policies of the domain. We demonstrate theoretically that, under a set of conditions to be satisfied, reusing such a set of eigen-policies allows us to bound the minimal expected gain received while learning a new policy. In general, Policy Reuse contributes to the overall goal of lifelong reinforcement learning, as (i) it incrementally builds a policy library; (ii) it provides a mechanism to reuse past policies; and (iii) it learns an abstract domain structure in terms of eigen-policies of the domain.

1. Introduction

Reinforcement Learning (RL) (Kaelbling, Littman, & Moore, 1996; Sutton & Barto, 1998) is a powerful technique to learn to solve different kind of tasks. The learning process is based on a trial and error process guided by reward signals received from the environment. The goal is to maximize the long term sum of the rewards obtained. Classical RL algorithms as Q-Learning (Watkins, 1989) rely on an intensive exploration of the action and state spaces. Due to the “curse of dimensionality” of such spaces in complex domains, solving a task typically requires and extensive interaction of the learning agent with the environment.

Although the cost (time, resources, etc.) of such a learning process may be very high, sometimes the task can be tackled and successfully solved (Tesauro, 1992; Stone, Sutton, & Kuhlmann, 2005). There have been many different efforts to address the complexity of the learning. To reuse the knowledge acquired in the current learning process when solving future problems, so the cost of future learning processes is reduced, is an appealing idea. In RL, several efforts have been done in this line, like the transfer of the value functions (Taylor, Stone, & Liu, 2005), the reuse of options (Sutton, Precup, & Singh, 1999) and the learning of hierarchies which modules can be used in the future (Dietterich, 2000).

©2005 AI Access Foundation. All rights reserved.
In this article, we report on Probabilistic Policy Reuse, a novel approach for transfer learning based on the reuse of similar action policies. It is based on our research in the related areas of Symbolic Plan Reuse (Veloso, 1994) and Extended Rapidly-exploring Random Trees (E-RRT) (Bruce & Veloso, 2002). Planning by analogical reasoning provides a method for symbolic plan reuse. However, when reusing a past plan, if a step becomes invalid to use in the new situation, the traditional reuse questions are: either (i) to resolve the locally failed step and direct the search to return back to another past plan step, or (ii) to completely abandon the past plan and re-plan from scratch from the failed step directly towards the goal. E-RRT solves this general reuse question by guiding a new plan probabilistically with a past plan. The past experience is effectively used as a bias in the new search, and thus solving the general reuse problem in a probabilistic manner.

Policy Reuse, as we introduce in this article, utilizes the past policies as a probabilistic bias where the learner faces three choices: the exploitation of the ongoing learned policy, the exploration of random unexplored actions, and the exploitation of past policies. We introduce $\pi - reuse$, a new exploration strategy to include a past policy as an exploration bias in a new learning process. From this strategy, a metric to estimate the similarity of past policies with respect to a new one is derived. This metric can be used to decide which policy is the most useful one to reuse from a set of policies. We contribute a new algorithm, PRQ-Learning (Policy Reuse in Q-Learning), that is able both to bias a new learning process with a set of past policies, and to use a similarity metric to decide which policies are more effective to reuse. Interestingly, a side-effect of Policy Reuse is its capability to identify classes of similar policies revealing a basis of eigen-policies of the domain. We demonstrate theoretically that, under a set of conditions to be satisfied, reusing such a set of eigen-policies allows us to bound the minimal expected gain received while learning a new policy. We contribute the PLPR algorithm (Policy Library through Policy Reuse), that allows to build such a basis of the domain.

This article is organized as follows. Section 2 summarizes relevant related work. Section 3 introduces Policy Reuse in the scope of Reinforcement Learning, and formalizes the concepts of task, domain, and gain. Section 4 defines the $\pi$-reuse exploration strategy and describes the results obtained with this exploration strategy in a large robot navigation domain, which is used in the different experiments along the paper. These results motivate the definition of a similarity metric among policies, as formalized in Section 5. We also describe the PRQ-Learning algorithm, and demonstrate how it can be efficiently used to reuse a set of past policies stored in a Policy Library. Section 6 presents the PLPR algorithm, and provides theoretical and empirical results that demonstrate the capability of the algorithm to build the basis of a domain as a set of eigen-policies. Lastly, Section 7 summarizes the main conclusions of this work.

2. Related Work

Policy Reuse 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. The exploration vs. exploitation problem defines whether to explore new or exploit the knowledge already acquired. The limits are defined by the random and the greedy strategies, and several can be found in between, as $\epsilon$-greedy and Boltzmann (Sutton
Probabilistic Policy Reuse

Directed exploration strategies memorize exploration-specific knowledge that is used for guiding the exploration search (Thrun, 1992). 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. These strategies only use knowledge obtained in the current learning process.

An interesting method to improve learning is by introducing knowledge in the exploration process. For instance, advice rules (Maclin, Shavlik, Torrey, Walker, & Wild, 2005) can be used to recommend that some action is preferred to another in a specified set of states. In this case, the source of the advice rules is the user, which is the source of exploration knowledge in many other approaches (Smart & Kaelbling, 2000). Different knowledge sources can be used, as a mentor, from which policies can be learned by imitation (Price & Boutilier, 2003). In the previous cases, as in Policy Reuse, the advice is about policies rather than Q values.

Transfer learning refers to the injection of knowledge from previously solved tasks. Memory guided exploration (Carroll, Peterson, & Owens, 2001) incorporates knowledge from a past policy in a new exploration process by weighting the Q values associated to the new and the past policy. However, that requires that the values of both Q functions are homogeneous and a perfect mapping between the past and the new Q function. The problem can be solved by weighting the probability of selecting each action, instead of the actual Q values (Dixon, Malak, & Khos, 2000). In any case, the choose of a correct weight decay to balance correctly the use of the past and the new policy relies on the designer.

The transfer of learning can be performed by transferring the Q-function instead of the policy. These algorithms directly transfer the Q values associated to a policy that solves a task to initialize a new Q function in a new learning process (Bowling & Veloso, 1999; Carroll & Peterson, 2002; Madden & Howley, 2004). However, if the source and target tasks are very heterogeneous, transfer learning also requires expert knowledge to perform the transfer (Taylor & Stone, 2005; Taylor et al., 2005). Then, expert knowledge is used to decide if the transfer has sense, and how to map actions and states from the source task to the target one. Value function transfer restricts the transfer learning capability to previous learning processes performed also through a value function. Furthermore, they do not focus on the case where several tasks have been previously solved and are susceptible to be reused.

A different way to introduce previous knowledge is by executing macro-actions or sub-policies. For instance, some algorithms use macro-actions to learn new action policies in Semi-Markov Decision Processes (SMDPs), as it is the case of TTree (Uther, 2002). Options can also be used in SMDPs (Sutton et al., 1999). They require the set of states from which they can be executed, an end condition and the behavior of the option. Such a behavior can be learned on line (Sutton, Precup, & Singh, 1998), as well as the other components of the option (Stolle & Precup, 2002). The transfer of learning can be performed by composing solutions of elemental sequential tasks (Singh, 1992). Hierarchical RL uses different abstraction levels to organize subtasks (Dietterich, 2000), and some approaches are able to learn such a hierarchy (Hengst, 2002). Somehow, what methods for learning hierarchies or options do is to learn the structure of the domain so it can be used in a long-life learning way. Some related algorithms are SKILL (Thrun & Schwartz, 1995), which tries to discover partially defined policies that arise in the context of multiple tasks in the
same domain, or L-Cut which tries to discovers subgoals and to learn subpolicies to achieve them (Şimşek, Wolfe, & Barto, 2005). Subpolicies can be reused when solving new tasks, sometimes suboptimally, although such a suboptimality can be bounded (Bowling & Veloso, 1999).

3. Policy Reuse in Reinforcement Learning

A Markov Decision Process (Puterman, 1994) is represented with a tuple \(<S, A, T, R>\), where \(S\) is the set of all possible states, \(A\) is the set of all possible actions, \(T\) is an unknown stochastic state transition function, \(T: S \times A \times S \rightarrow \mathbb{R}\), and \(R\) is an unknown stochastic reward function, \(R: S \times A \rightarrow \mathbb{R}\). We focus in RL domains where different tasks can be solved. MDP’s formalism is not expressive enough to represent all the concepts involved in knowledge transfer (Sherstov & Stone, 2005), so domain and task are defined separately to handle different tasks executed in the same domain. In our case, we introduce a task as a specific reward function, but the other concepts, \(S\), \(A\) and \(T\) stay constant for all the tasks in the same domain. We characterize a domain, \(D\), as a tuple \(<S, A, T>\). We define a task, \(\Omega\), as a tuple \(<D, R_\Omega>\), where \(D\) is a domain as defined before, and \(R_\Omega\) is the stochastic and unknown reward function. Both definitions are formalized next:

**Definition 1.** A Domain \(D\) is defined as a tuple \(<S, A, T>\), where \(S\) is the set of all possible states; \(A\) is the set of all possible actions; and \(T\) is a state transition function, \(T: S \times A \times S \rightarrow \mathbb{R}\).

**Definition 2.** A task \(\Omega\) is defined as a tuple \(<D, R_\Omega>\), where \(D\) is a domain; and \(R_\Omega\) is the reward function, \(R: S \times A \rightarrow \mathbb{R}\).

In this work we assume that we are solving an episodic task with absorbing goal states. Thus, if \(s_i\) is a goal state, the probability of transiting to the same state is 1 (\(T(s_i, a, s_i) = 1\)), the probability of transition to a different state is 0 (\(T(s_i, a, s_j) = 0\) for \(s_i \neq s_j\)), and the immediate reward is 0 (\(R(s_i, a) = 0\), for all \(a \in A\)). A trial or episode starts by locating the learning agent in a random position in the environment. Each episode 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 episode, say \(W\), as defined in equation 1:

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

where \(\gamma\) (\(0 \leq \gamma \leq 1\)) 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.

We define an action policy, \(\Pi\), as a function \(\Pi: S \rightarrow A\). If the action policy was created to solve a defined task, \(\Omega\), the action policy is called \(\Pi_\Omega\). The gain, or average expected reward, received when executing an action policy \(\Pi\) in the task \(\Omega\) is called \(W^\Pi_{\Omega}\). Lastly, an optimal action policy for solving the task \(\Omega\) is called \(\Pi^*_\Omega\). The action policy \(\Pi^*_\Omega\) is optimal

---

1. The concepts of domain and task provided in (Sherstov & Stone, 2005) are more general than the required in this paper. A mapping from one to the other can be performed easily. The concept of Task Distribution (Bernstein, 1999) also tries to generalize MDPs to multiple tasks by providing each task with a different MDP but also associating a probability of execution to each different task.
if $W_{\Omega}^{\Pi^*} \geq W^H_{\Omega}$, for all policy $\Pi$ in the space of all possible policies when $K \to \infty$. Action policies can be represented using the action-value function, $Q^H(s, a)$, that defines for each state $s \in S$, $a \in A$, the expected reward that will be obtained if the agent start to act from $s$, executing $a$, and after it follows the policy $\Pi$. So, the RL problem is translated to learning the function $Q^H(s, a)$. This learning can be performed using different algorithms, as Q-Learning (Watkins, 1989).

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 $\Omega_i = \langle D, R_{\Omega_i} \rangle$, i.e. all policies solve a task in the same domain.

Previous definition does not restrict the 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 need to solve the task $\Omega$, i.e. learn $\Pi^*_\Omega$
- We have previously solved the set of tasks $\{\Omega_1, \ldots, \Omega_n\}$ so we have a Policy Library composed of the $n$ policies that solve them respectively, say $L = \{\Pi_1, \ldots, \Pi_n\}$
- How can we use the policy library, $L$, to learn the new policy, $\Pi^*_\Omega$?

The answer to this question requires an exploration strategy able to bias the exploration process with the policies stored in 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 requires an efficient method to construct the Policy Library. In the following sections we describe our approach for each of the above components of Policy Reuse.

### 4. Reusing a Past Policy

The goal of this section is to describe how one policy, which solves a task can be used to bias the learning of the action policy of another similar task. Then, the scope of this section is the following:

- We need to solve the task $\Omega$, i.e. learn $\Pi^*_\Omega$.
- We have a Policy Library, say $L = \{\Pi_1, \ldots, \Pi_n\}$
- Let’s assume that there is a supervisor who, given $\Omega$, tells us which is the most similar policy, say $\Pi_{past}$, to $\Pi^*_{\Omega}$. Thus, we know that the policy to reuse is $\Pi_{past}$.

Thus, in this section we assume that it exists a supervisor who provides a policy similar to the one that we are trying to solve. A discussion on how similarities between tasks and their respective policies can be computed, and how to automatically estimate the policy to reuse, will be introduced in Section 4.2.4.
4.1 The $\pi$-reuse Exploration Strategy

The $\pi$-reuse strategy is an exploration strategy able to bias a new learning process with a past policy. We denote the past policy with $\Pi_{\text{past}}$, and the one we are currently learning with $\Pi_{\text{new}}$. We assume that we are using a direct RL method to learn the action policy, so we are learning its 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 (Watkins, 1989).

The goal of $\pi$-reuse is to balance random exploration, exploitation of the old policy, and exploitation of the new policy, which is being learned currently. It is summarized in Equation 2.

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

The equation shows that the $\pi$-reuse strategy follows the past policy with a probability of $\psi$. 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 $\epsilon$-greedy strategy. Table 1 shows a procedure describing the $\pi$-reuse strategy integrated with the Q-Learning algorithm. The procedure receives the past policy, $\pi_{\text{past}}$, the number of episodes, $K$, the maximum number of steps per episode, $H$, and the $\psi$ parameter. An additional $\nu$ parameter has been added to decay the value of $\psi$ in each step of the episode. The procedure outputs the $Q$ function, the policy, and the average gain obtained in the execution, $W$, which will be play an important role in the next sections. The variable $\psi_h$ stores the value of $\nu$ in each step episode.

\[
\begin{align*}
\text{\pi-reuse (}\Pi_{\text{past}}, K, H, \psi, \nu) & \quad \text{ Initialize } Q^{\Pi_{\text{new}}}(s, a) = 0, \forall s \in S, a \in A \\
& \quad \text{For } k = 1 \text{ to } K \\
& \quad \quad \text{ Set the initial state, } s, \text{ randomly.} \\
& \quad \quad \text{ Set } \psi_1 \leftarrow \psi \\
& \quad \quad \text{ for } h = 1 \text{ to } H \\
& \quad \quad \quad \text{ With a probability of } \psi_h, a = \Pi_{\text{past}}(s) \\
& \quad \quad \quad \text{ With a probability of } 1 - \psi_h, a = \epsilon - \text{greedy}(\Pi_{\text{new}}(s)) \\
& \quad \quad \quad \text{ Receive current state } s', \text{ and reward, } r_{k,h} \\
& \quad \quad \quad \text{ Update } Q^{\Pi_{\text{new}}}(s, a), \text{ and therefore, } \Pi_{\text{new}}, \text{ using the Q-Learning update function:} \\
& \quad \quad \quad Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')] \\
& \quad \quad \quad \text{ Set } \psi_{h+1} \leftarrow \psi_h \nu \\
& \quad \quad \quad \text{ Set } s \leftarrow s' \\
& \quad \quad \text{ W } \leftarrow \sum_{k=0}^{K} \sum_{h=0}^{H} \gamma^h r_{k,h} \\
& \quad \quad \text{ Return } W, Q^{\Pi_{\text{new}}}(s, a) \text{ and } \Pi_{\text{new}} \\
\end{align*}
\]

Table 1: $\pi$-reuse Exploration Strategy.

Thus, there are three probabilities involved: the probability of exploiting the past policy, i.e., $\psi_h$, the probability of using the current policy, i.e., $\epsilon (1 - \psi_h)$, and the probability of acting randomly, i.e., $(1 - \epsilon)(1 - \psi_h)$. These probabilities are shown in Figure 1, for input values of $H = 100$, $\psi = 1$ and $\nu = 0.95$. In this case the $\epsilon$ parameter is set to $1 - \psi_h$ in each step.
With this parameter setting, the exploration is biased with the past policy mainly in the initial steps of the episode. Assigning to $\epsilon$ the value of $(1 - \psi_h)$ makes the strategy very greedy in the final steps of each episode, given that we assume that the last steps are the ones that are learned faster (since rewards are also propagated fast from the goal). The figure shows that in the initial steps of each episode, the past policy is exploited. As the number of steps increases, the probabilities of exploiting the new policy and acting randomly increases. In the final steps of the episode, only the new policy is exploited. The transition from exploiting the past policy and exploiting the new one depends on the $\nu$ parameter. If this parameter is low, the transition occurs in the initial steps, while if it is high, the transition is delayed.

### 4.2 Experiments

In this section, we describe the experiments performed to demonstrate the usefulness of the $\pi$-reuse exploration strategy. But first, we describe the navigation domain used.

#### 4.2.1 Robot Navigation Domain

This domain consists of a robot moving inside of an office area, as shown in Figure 2. The environment is represented by walls, free positions and goal areas, all of them of size $1 \times 1$. The whole domain is $N \times M$ (24 $\times$ 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 $24 \times 21$ regions)\(^2\). Furthermore, the robot has an obstacle avoidance system that blocks the execution of

---

\(^2\) Different methods for function approximation has been successfully applied on this this domain (Fernández & Borrajo, 2002). We have simplified the state space representation to a uniform discretization to focus on the study of Policy Reuse.
actions that would crash it into a wall. 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.

Figure 2 shows 6 different tasks, $\Omega_1$, $\Omega_2$, $\Omega_3$, $\Omega_4$, $\Omega_5$ and $\Omega$, given that the goal states, and therefore, the reward functions, are different. All these tasks are used in the experiments described in next sections.

![Figure 2: Office Domain.](image)

We choose the robot navigation domain for experimentation because it has been widely used in transfer learning papers, since the early works on structure learning (Thrun & Schwartz, 1995), until recent works on transfer learning (Madden & Howley, 2004; Sherstov & Stone, 2005) and other related areas (Price & Boutilier, 2003; Stolle & Precup, 2002). 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 (Taylor et al., 2005; Taylor & Stone, 2005). However, that mapping requires a considerable amount of expert knowledge. Our future research line is to analize how our policy reuse algorithm can be extended to include such a knowledge.

**4.2.2 Description of the Learning and Test Curves**

In the following subsections, we describe the experimental results of applying different exploration strategies for learning the task $\Omega$, shown in Figure 2(f). For each of these strategies (and parameter settings), 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 $\gamma = 0.95$ and $\alpha = 0.05$, which empirically have demonstrated to be accurate for learning in this domain. \(^3\)

A learning performance consists of executing $K = 2000$ episodes. Each episode consists on 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 episode or trial number. The $y$ axis represents the average gain obtained. Thus, a value of 0.2 for the episode 200 means that the average gain obtained in the 200 first episodes has been 0.2.

The test curve represents the evolution of the performance of the policy while it is being learned. Each 100 episodes 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 of 1000 episodes where the robot follows a completely greedy strategy. Thus, the $x$ axis shows the learning episode 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 episodes.

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.

4.2.3 Learning from Scratch

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 $\epsilon$-greedy, for an initial value of $\epsilon = 0$, which is incremented by 0.0005 in each episode. Lastly, Boltzmann strategy has been used, initializing $\tau = 0$, and increasing it in 5 in each learning episode.

Figure 3: Learning and test evolution when learning from scratch.

Figure 3(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 main 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 produces very different results in different executions. The curve obtained by the Boltzmann strategy does not offer many improvements but a reduction in the standard deviation. However, the $\epsilon$-greedy strategy seems to compute an accurate policy in the initial episodes, and obtain the highest average gain at the end of the learning.

The random strategy and $\epsilon$-greedy outperforms the other strategies in the test curve shown in Figure 3(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, demonstrating that a higher exploration at the initial steps of the learning results in more accurate policies in test.

4.2.4 Reusing the Past Policy Following $\pi$-reuse

Figure 4(a) shows the learning curves of different learning processes performed with the $\pi$-reuse exploration strategy. In each of them, a different policy has been reused. 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 $\pi$-reuse exploration strategy are the ones used to obtain the evolution of the probabilities described in Figure 1 ($\psi = 1$, $\nu = 0.95$ and $\epsilon = 1 - \psi h$), which have demonstrated empirically to be accurate. 4 We distinguish three different cases. In the first one, the policy reused is $\Pi_5$ ($\Pi_{past} = \Pi_5$), whose associated task $(\Omega_5)$ has its goal into the same room as the goal of $\Omega$. In the second case, $\Pi_{past} = \Pi_{1}$, so their respective goals are in different rooms. However, their optimal policies could be the same for all the domain except for the rooms where the respective goals are located. In the last two cases, $\Pi_{past} = \Pi_2$ and $\Pi_3$ respectively, whose associated tasks $(\Omega_2$ and $\Omega_3$) are very different when compared to $\Omega$.

![Learning Curve](a)

![Test Curve](b)

Figure 4: Learning and test evolution when following the exploration strategy $\pi$-reuse.

Figure 4(a) shows how, when biasing the exploration process for learning the task $\Omega$ with the policies $\Pi_1$ and $\Pi_5$, the obtained gain increases dramatically within the first few episodes of the execution. For instance, when reusing $\Pi_1$, in only 100 iterations the average gain is higher than 0.15, and after 400 iterations the value stays around 0.2. When reusing

4. The parameter setting has been chosen after an informal experimentation, so maybe it is not not optimal, and different values 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.
Π₅, 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. The behavior of the test curves is also very good in both cases, showing that in only 400 iterations, a gain higher than 0.3 is obtained with a very low deviation. These results demonstrate that reusing similar past policies produces a significant improvement over exploration strategies that learn from scratch.

However, when the learning is biased with a very different policy, as Π₂ and Π₃, the average gain shown in Figure 4(a) is below 0.05, so the learning process is even worse than when learning from scratch. Their test curves present a better behavior. In both cases there is an inflexion in the test curve, obtaining, at the end of the 2000 episodes, a similar performance than unbiased strategies. The inflexion is due to the learning of an initial path to the goal. However, in this case the standard deviation is very high, demonstrating that the inflexion may occur in very different moments of the learning process.

The results show that reusing a past policy provides a bias in the exploration process which could speed up the learning when compared when learning from scratch. The improvement depends on whether the reused policy solves a task similar to the one we are currently learning. However, a similarity metric between tasks is hard to define even for a well structured domain. For instance, in a office navigation problem, Manhattan distance between the goals could be easily used. In such a case, the distance between tasks Ω and Ω₅ is 3, and between Ω and Ω₁ is 4. However, this similarity metric could fail because of the walls (that generate borders in the value function, which are very typicall in most of the RL domains (Munos & Moore, 2002)). Furthermore, it requires to discover where the goal is which could be unpractical because of the diversity of goal representations in different domains and/or tasks and the exploration process required to learn them.

However, similarity between policies can be easily defined. For instance, the results in Figure 4(a) shows that the learning curves provide us with a metric of 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 one could be used to rank the similarity of the past policies with respect to the new one. In this case, the most similar policy to Π₅ is Π₅, followed by Π₁, Π₂ and Π₃.

Furthermore, the estimations above can be computed very fast, as Figure 5 demonstrates. The figure zooms in on the initial 100 episodes of Figure 4(a). The figure shows that in only 25 episodes, the gain of reusing the policy Π₅ significantly outperforms the gain of reusing the other policies. Thus, in a total of 100 episodes (25 for each policy), the most similar policy, and therefore, the best policy to reuse, can be computed. These ideas are described in the following section, where a formal description of the similarity metric is introduced.

5. Reusing a Library of Policies

This section describes the PRQ-learning algorithm, which allows to efficiently reuse the policies stored in a Policy Library. Firstly, we describe a similarity metric, based on above results, that estimates how useful to reuse a policy is for learning a new one.
5.1 A Similarity Metric Between Policies

The exploration strategy \(\pi\)-reuse, as defined in Table 1, returns the learned policy, \(\Pi_{\text{new}}\), and the average gain obtained in its learning process, \(W\). Let’s call \(W_i\) the gain obtained while executing the \(\pi\)-reuse exploration strategy, reusing the past policy \(\Pi_i\). \(\Pi^*_\Omega\) is the optimal action policy for solving the task \(\Omega\), and \(W^*_\Omega\) is the gain obtained when using the optimal policy, \(\Pi^*_\Omega\), to solve \(\Omega\). Therefore, \(W^*_\Omega\) is the maximum gain that can be obtained in \(\Omega\). Then, we can use the difference between \(W^*_\Omega\) and \(W_i\) to measure how useful to reuse the policy \(\Pi_i\) is to learn to solve the new task. Next definitions formalize these ideas.

Definition 4. Given a policy \(\Pi_i\) that solves a task \(\Omega_i =< \mathcal{D}, R_i >\) and a new task \(\Omega =< \mathcal{D}, R_\Omega >\); the Reuse Gain of the policy \(\Pi_i\) on the task \(\Omega\), say \(W_i\), is the gain obtained when applying the \(\pi\)-reuse exploration strategy with the policy \(\Pi_i\) to learn the policy \(\Pi\).

Definition 5. Given a policy \(\Pi_i\) that solves a task \(\Omega_i =< \mathcal{D}, R_i >\), a new task \(\Omega =< \mathcal{D}, R_\Omega >\), and its respective optimal policy, \(\Pi^*\); the Reuse Distance from \(\Pi_i\) to \(\Pi^*\), say \(d_{\rightarrow}(\Pi_i, \Pi^*)\), is defined by equation 3, where \(W^*_\Omega\) is the gain obtained when solving the task \(\Omega\) with the optimal policy, \(\Pi^*\), and \(W_i\) is the Reuse Gain of the policy \(\Pi_i\) on the task \(\Omega\).

\[
d_{\rightarrow}(\Pi_i, \Pi^*) = W^*_\Omega - W_i
\]  

The arrow in the distance metric defines the direction of the similarity metric, given that this metric may not be symmetric and therefore \(d_{\rightarrow}(\Pi_i, \Pi^*)\) may be different to \(d_{\rightarrow}(\Pi^*, \Pi_i)\). Symmetry depends on the domain and the task.

Then, the most useful policy to reuse, say \(\Pi_k\) from a Library Policy, \(L = \{\Pi_1, \ldots, \Pi_n\}\), is the one that minimize the Reuse Distance, as defined in equation 4:

\[
\Pi_k = \arg_{\Pi \in L} \min d_{\rightarrow}(\Pi_i, \Pi^*) = \arg_{\Pi \in L} \min(W^*_\Omega - W_i), i = 1, \ldots, n
\]  

\(W^*_\Omega\) is independent of \(i\), so the previous equation is equivalent to obtain the policy which provides a higher Reuse Gain:

\[
\Pi_k = \arg_{\Pi \in L} \max(W_i), i = 1, \ldots, n
\]
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.

5.2 PRQ-Learning Algorithm

The goal of the PRQ-learning algorithm is to solve a task $\Omega$, i.e. to learn an action policy $\Pi_\Omega$. 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’s call $W_i$ the Reuse Gain of the policy $\Pi_i$ on the task $\Omega$. Also, let’s call $W_\Omega$ the average reward that is received when following the policy $\Pi_\Omega$ greedily. The solution proposed is to follow a softmax strategy, using the values $W_\Omega$ and $W_i$, as defined in equation (6), where a temperature parameter $\tau$ is included. Notice that this value is also computed for $\Pi_0$, which we assume to be $\Pi_\Omega$.

$$P(\Pi_j) = \frac{e^{\tau W_j}}{\sum_{\rho=0}^{n} e^{\tau W_\rho}}$$  (6)

Previous equation provides a simple way to decide whether to exploit the past policies or the new one. A high value of the reuse gains ($W_i$) will only be comp

The answer to the second question depends on the selected policy. If the policy chosen is $\Pi_\Omega$, a completely greedy strategy is followed. However, if the policy chosen is $\Pi_i$ (for $i = 1, \ldots, n$), the $\pi$-reuse action selection strategy, defined in previous section, is followed. In this way, the Reuse Distance among the past policies and the new one can be estimated on-line with the learning of the new policy. Thus, the values required in Equation 6 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 receives: a new task to solve, $\Omega$; the policy library $L$; the temperature parameter of the softmax policy selection equation, $\tau$, and a decay parameter, $\delta \tau$; and a set of previously defined parameters: $K, H, \psi, v, \gamma, \alpha$.

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, 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. However, while new updates are performed over the Q function, it becomes more accurate, and will receive high 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.
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. $Q_{\Omega}(s,a) = 0$, $\forall s \in S, a \in A$
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 6):

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

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.

5.3 Experiments

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 and/or useful to reuse. 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 learned currently can be successfully achieved.
We use the PRQ-Learning algorithm for learning the task $\Omega$, defined in Figure 2(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 $\Omega_2$, $\Omega_3$ and $\Omega_4$, defined in Figure 2(b), (c) and (d) respectively, where 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, $\Pi_1$ is added, so $L_2 = \{\Pi_1, \Pi_2, \Pi_3, \Pi_4\}$. The third case uses the Policy Library $L_3 = \{\Pi_2, \Pi_3, \Pi_4, \Pi_5\}$.

![Learning Curve](a)

![Test Curve](b)

Figure 6: Learning and test curves when learning the task of Figure 2(f) reusing different sets of policies.

The PRQ-Learning algorithm is executed for the three cases. The learning curves are shown in Figure 6(a). The parameters used are the same used in Section 4.2. The only new parameters are the ones of the Boltzmann policy selection strategy, $\tau = 0$, and $\Delta \tau = 0.05$, obtained empirically.

Figure 6(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. For instance, when reusing $\Pi_1$ and $\Pi_5$, the average gain is greater than 0.1 in only 500 iterations, and more than 0.25 at the end of the episode. On the other hand, the learning curve when no similar policy is available 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.

Figure 6(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 episodes. That is a strong improvement over the strategies that learn from scratch reported in Figure 3.

The good results obtained when reusing past similar policies can be easily understood if we look in the Figure 7, which reports about the learning process when reusing the Policy Library $L_3 = \{\Pi_5, \Pi_2, \Pi_3, \Pi_4\}$. Figure 7(a) shows the evolution of the Reuse Gain computed for each policy involved, $W_5$, $W_2$, $W_3$, $W_4$, and the gain $W_{\Omega}$. On the $x$ axis, the number of episodes is shown, while the $y$ axis shows the gains. Initially, the Reuse Gain of all the
policies is set to 0. After a few episodes, $W_2$, $W_3$, and $W_4$ stabilize below 0.05. However, $W_5$ increases up to 0.15. These values demonstrate that the most similar policy ($\Pi_5$) is correctly computed. The gain of the new policy, $W_\Omega$, starts to increase around iteration 100, achieving a value higher than 0.3 by iteration 500, demonstrating that the new policy is very accurate.

The values of the Reuse Gain 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 6, and the parameters introduced above (initial $\tau = 0$, and $\Delta\tau = 0.05$). Figure 7(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, the probability of $\Pi_5$ grows. For the other past policies, the probability decreases down to 0. The probability of the new policy also increases, and after 400 iterations, its bigger than the rest. After a few iterations more, it achieves the value of 1, given that its gain is the higher, as shown in Figure 7(a).

Figure 7(b) demonstrates how the balance between exploiting the past policies or the new one is achieved. It shows how in the initial episodes, the algorithm choses to reuse the past policies to find the most similar. Then, it reuses the most similar policy until the new policy is leaned and improves the result of reusing any past policy.

As a conclusion, we can say that the PRQ-learning algorithm has demonstrated to successfully reuse a predefined set of policies. The remaining problem is that it requires the existence of such a set of policies. Next section focuses on how to build a library of policies.


This section describes 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. Notice that we are assuming that the tasks that the algorithm will be asked to solve are unknown a priori, and are given in a sequential way. Otherwise, a method to learn them in parallel could be applied (Ollington & Vamplew, 2005).
6.1 The PLPR Algorithm

The algorithm works as follows. Initially, the Policy Library is empty, $PL = \emptyset$. Then, the first task, say $\Omega_1$, needs to be solved, so the first policy, say $\Pi_1$, is learned. To learn the first policy, any exploration strategy could be used but the policy reuse strategy $\pi$-reuse, given that there is not any available policy to reuse. $\Pi_1$ is added to the Policy Library, so $PL = \{\Pi_1\}$. When a second task needs to be solved, the PRQ-Learning algorithm is applied, reusing $\Pi_1$. Thus, $\Pi_2$ is learned. Then, we need to decide whether to add $\Pi_2$ to the Policy Library or not. This decision is based on how similar $\Pi_1$ is to $\Pi_2$, following the Reuse Distance defined in equation 3, instantiated in equation 7 for this case. In the equation, $W_2$ is the average gain obtained when following $\Pi_2$ greedily, and $W_1$ is the Reuse Gain of $\Pi_1$ on task $\Omega_2$. Both values are computed in the execution of the PRQ-Learning algorithm, so no additional computations are required.

$$d_{\rightarrow}(\Pi_1, \Pi_2) = W_2 - W_1$$  \hspace{1cm} (7)

As defined in the previous section, this distance metric estimates how similar $\Pi_1$ is to $\Pi_2$. In our case, if $\Pi_1$ is very similar to $\Pi_2$, i.e. $d_{\rightarrow}(\Pi_1, \Pi_2)$ is close to 0, to include the second policy in the library is unnecessary. However, if the distance is large, $\Pi_2$ is included. Therefore, we can introduce a new concept, $\delta$-similarity, as follows.

**Definition 6.** Given a policy, $\Pi_i$ that solves a task $\Omega_i = \langle D, R_i \rangle$, a new task $\Omega = \langle D, R_\Omega \rangle$, and its respective optimal policy, $\Pi$. $\Pi$ is $\delta$-similar to $\Pi_i$ (for $0 \leq \delta \leq 1$) if $W_i > \delta W^*_\Omega$, where $W_i$ is the Reuse Gain of $\Pi_i$ on task $\Omega$ and $W^*_\Omega$ is the average gain obtained in $\Omega$ when an optimal policy is followed.

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

**Definition 7.** Given a Policy Library, $L = \{\Pi_1, \ldots, \Pi_n\}$ in a domain $D$, a new task $\Omega = \langle D, R_\Omega \rangle$, and its respective optimal policy, $\Pi$. $\Pi$ is $\delta$-similar with respect to $L$ iff $\exists \Pi_i$ such as $\Pi$ is $\delta$-similar to $\Pi_i$, for $i = 1, \ldots, n$.

Thus, if we know that a policy $\Pi$ is $\delta$-similar with respect to a Policy Library $L$, we know that the policy $\Pi$ can be easily learned by reusing the policies in $L$.

The PLPR algorithm is described in Table 3. It is executed each time that a new task needs to be solved. It inputs the Policy Library and the new task to solve, and outputs the learned policy and the updated Policy Library.

Equation 8 is the update equation for the Policy Library, derived from equation 7. It requires the computation of the most similar policy, which is the policy $\Pi_j$ such as $j = \arg \max W_i$, for $i = 1, \ldots, n$. The gain obtained by reusing such a policy is called $W_{\max}$. The new policy learned is inserted in the library if $W_{\max}$ is lower than $\delta$ times the gain obtained by using the new policy ($W_\Omega$), where $\delta \in [0, 1]$ defines the similarity threshold, i.e. whether the new policy is $\delta$-similar with respect to the Policy Library.

The parameter $\delta$ has an important role. If it receives a value of 0, the Policy Library stores only the first policy learned, given that the average gain obtained by reusing it will
PLPR Algorithm

• Given:
  1. A Policy Library, \( L \), composed of \( n \) policies, \( \{\Pi_1, \ldots, \Pi_n\} \)
  2. A new task \( \Omega \) we want to solve
  3. A \( \delta \) parameter

• Execute the PRQ-Learning algorithm, using \( L \) as the set of past policies. Receive from this execution \( \Pi_\Omega, W_\Omega \) and \( W_{max} \), where:
  - \( \Pi_\Omega \) is the learned policy
  - \( W_\Omega \) is the average gain obtained when the policy \( \Pi_\Omega \) was followed
  - \( W_{max} = \max W_i \), for \( i = 1, \ldots, n \)

• Update PL using the following equation:

\[
L = \begin{cases} 
L \cup \{\Pi_\Omega\} & \text{if } W_{max} < \delta W_\Omega \\
L & \text{otherwise}
\end{cases}
\]  

Table 3: PLPR Algorithm.

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, due to the fact that \( W_{max} < W_\Omega \), given that \( W_\Omega \) is maximum if the optimal policy has been learned. Different values in the range \((0, 1)\) provide different sizes of the library, as will be demonstrated in the experiments. Thus, \( \delta \) defines the size, and therefore the resolution, of the library.

6.2 Eigen Analysis of Policy Reuse

The PLPR algorithm has an interesting side-effect in terms of learning the structure of the domain. Notice that the Policy Library is initialized to empty, and a new policy is included only if it is different enough with respect to the previously stored ones, depending on the threshold \( \delta \). When the policies stored is fully representative of the domain, no more policies are stored. Thus, the obtained library can be considered as the Basis-Library of the domain, and the stored policies can be considered as the eigen-policies of such domain. In the following, we provide some definitions that formalize these concepts. We also describe: (i) a theorem that defines the conditions under which the PLPR algorithm builds a \( \delta \)-Basis-Library of the domain (Theorems 1 and 2); and (ii) a theorem that demonstrates that any task can be efficiently learned by reusing the \( \delta \)-Basis-Library of the domain, defining a lower bound of the expected reward that will be received when reusing the Basis-Library for learning the new task (Theorem 3).

Definition 8. A Policy Library, \( L = \{\Pi_1, \ldots, \Pi_n\} \) in a domain \( \mathcal{D} \) is a \( \delta \)-Basis-Library of the domain \( \mathcal{D} \) iff: (i) \( \nexists \Pi_i \in L \), such as \( \Pi_i \) is \( \delta \)-similar with respect to \( L - \Pi_i \); and (ii) no policy \( \Pi \) in the space of all the possible policies in \( \mathcal{D} \) is \( \delta \)-similar with respect to \( L \).

Definition 9. Given a \( \delta \)-Basis-Library, \( L = \{\Pi_1, \ldots, \Pi_n\} \) in a domain \( \mathcal{D} \), a new task \( \Omega = < \mathcal{D}, R_\Omega > \), each policy \( \Pi \in L \) is called a \( \delta \)-Eigen-Policy of the domain \( \mathcal{D} \) in \( L \).

The proper computation of the Reuse Gain for each past policy in the PRQ-Learning algorithm has a very important role, since it allows us to compute the most similar policy,
its reuse distance and therefore, to decide whether to add the new policy to the Policy Library or not. If the reuse gain is not correctly computed, the basis library may not be either. Thus, we introduce a new concept that measures how accurate the estimation of the reuse gain is.

**Definition 10.** Given a Policy Library, $L = \{\Pi_1, \ldots, \Pi_n\}$ in a domain $D$, and a new task $\Omega = < D, R_\Omega >$. Let’s assume that the PRQ-Learning algorithm is executed, so it outputs the new policy $\Pi_\Omega$, the estimation of the optimal gain $\hat{W}_{\Pi_\Omega}$, and the estimation of the Reuse Gain of the most similar policy, say $\hat{W}_{\max}$. We say that the PRQ-Learning algorithm has been Properly Executed with a confidence factor $f$ ($0 \leq f \leq 1$), if $\Pi_\Omega$ is optimal to solve the task $\Omega$, and the error in the estimation of both parameters is lower than a factor of $f$, i.e.:

$$
\begin{align*}
\hat{W}_{\max} & \geq f W_{\max} \\
\hat{W}_{\Pi_\Omega} & \geq f W_{\Pi_\Omega} \\
f \hat{W}_{\Pi_\Omega} & \leq W_{\Pi_\Omega} 
\end{align*}
$$

(9)

where $W_{\max}$ is the actual value of the Reuse Gain of the most similar policy and $W_{\Pi_\Omega}$ is the actual gain of the obtained policy.

Thus, if we say that the PRQ-Learning algorithm has been Properly Executed with a confidence of 0.95, we can say, for instance, that the estimated Reuse Gain, $\hat{W}_{\max}$ of the most similar policy, has a maximum deviation over the actual Reuse Gain of a 5%. The proper execution of the algorithm depends on how accurate the parameters selection is. Such a parameter selection depends on the domain and the task, so no general guidelines can be provided.  

Previous definition let us to enumerate the conditions that makes the PLPR algorithm to build a $\delta$-Basis-Library, as described in the following theorem.

**Theorem 1.** The PLPR algorithm builds a $\delta$-Basis-Library if (i) the PRQ-Learning algorithm is Properly Executed with a confidence of 1; (ii) the Reuse Distance is symmetric; and (iii) the PLPR algorithm is executed infinite times over random tasks.

**Proof.** The proper execution of the PRQ-Learning algorithm ensures that the similarity metric, and all the derived concepts, are correctly computed. The first condition of the definition of $\delta$-Basis-Library can be demonstrated by induction. The basis case is when the library is composed of only one policy, given that no policy is $\delta$-similar with respect to an empty library. The inductive hypothesis is that a Policy Library $L_n = \{\Pi_1, \ldots, \Pi_n\}$ is a $\delta$-Basis-Library. Lastly, the inductive step is that the library $L_{n+1} = \{\Pi_1, \ldots, \Pi_n, \Pi_{n+1}\}$ is also a $\delta$-Basis-Library. If the PLPR algorithm has been followed to insert $\Pi_{n+1}$ in $L$, we ensure that $\Pi_{n+1}$ is not $\delta$-similar with respect to $L$, given that is the condition to insert a new policy in the library, as derived from equation 7. Furthermore, $\Pi_i$ (for $i = 1, \ldots, n$) is not $\delta$-similar with respect to $L_{n+1} - \Pi_i$, given that (i) $\Pi_i$ is not $\delta$-similar with respect to $L_n - \Pi_i$ (for inductive hypothesis); and (ii) $\Pi_i$ is not $\delta$-similar to $\Pi_{n+1}$ because the reuse distance is symmetric (by second condition of the theorem), and that ensures that if $\Pi_i$ is

---

5. The concept of proper execution of the PRQ-Learning algorithm arises again the exploration and exploitation trade-off. On the one side, if we focus on a very accurate estimation of the Reuse Gain, exploration may be wasted in terms of maximizing the average gain obtained while learning. On the other side, if we focus on maximizing the average gain of the current episode, the reuse gain of the past policies could be wrong, and therefore, the reuse distance and all the derived concepts too.
not $\delta$-similar to $\Pi_{n+1}$, then $\Pi_{n+1}$ is not $\delta$-similar to $\Pi_i$. Lastly, the second condition of the definition of $\delta$-Eigen-Policy becomes true if the algorithm is executed infinite times, which is satisfied by the third condition of the theorem.

The achievement of the conditions of the theorem depends on several factors. As introduced in Section 5.1, symmetry in the Reuse Distance depends on the task and the domain. The proper execution of the PRQ-Learning algorithm also depends on the selection of the correct parameters for each domain. However, although previous theorem requires that the PRQ-Learning algorithm to be properly executed with a confidence of 1, a generalized result can be easily derived when the confidence is under 1, say $f$, as the following theorem claims.

**Theorem 2.** The $\text{PLR}$ algorithm builds a $(2f\delta)$-Basis-Library if (i) the PRQ-Learning algorithm is Properly Executed with a confidence of $f$; (ii) the Reuse Distance is symmetric; and (iii) the PLPR algorithm is executed infinite times over random tasks.

**Proof.** The proof of this theorem only requires a small consideration over the inductive step of the proof of the previous theorem, where a policy $\Pi_{n+1}$ is inserted in the $\delta$-Eigen-Policy $L_n = \{\Pi_1, \ldots, \Pi_n\}$ following the PLR algorithm. The policy is added only if it is not $\delta$-similar with respect to $L_n$. In that case, if the PRQ-Learning algorithm has been properly executed with a confidence of $f$, we can only ensure that the policy $\Pi_{n+1}$ is not $(2f\delta)$-similar with respect to $L_n$, because of the error in the estimation of the gains (reuse gain and optimal gain) in the execution of the PRQ-Learning algorithm.

Lastly, we define a lower bound of the learning gain that is obtained when reusing a $\delta$-Basis-Library to solve a new task.

**Theorem 3.** Given a $\delta$-Basis-Library, $L = \{\Pi_1, \ldots, \Pi_n\}$ of a domain $D$, and a new task $\Omega = < D, R_\Omega >$. The average gain obtained, say $W_\Omega$, when learning a new policy $\Pi_\Omega$ to solve the task $\Omega$ by properly executing the PRQ-Learning algorithm over $\Omega$ reusing $L$ with a confidence factor of $f$ is at least $f\delta$ times the optimal gain for such a task, $W_\Omega^*$, i.e.

$$W_\Omega > f\delta W_\Omega^*$$  \hspace{1cm} (10)

**Proof.** When executing the PRQ-Learning properly, we reuse all the past policies, obtaining an estimation of their reuse gain. In the definition of Proper Execution of the PRQ-Learning algorithm, the gain generated by the most similar task, say $\Pi_i$, was called $\hat{W}_{\max}$, which is an estimation of the real one. In the worst case, the gain obtained in the execution of the PRQ-Learning algorithm is provided only by the most similar policy, $\Pi_i$, and the gain obtained by reusing any other different policy is 0, i.e. $W_j = 0, \forall \Pi_j \neq \Pi_i$. By the definition of $\delta$-Basis-Library we know that every policy $\Pi$ in the domain $D$ is not $\delta$-similar with respect to $L$. Thus, the most similar policy in $L$, $\Pi_i$ is such that its Reuse Gain, $W_{\max}$ satisfies $W_{\max} > \delta W_\Omega^*$ (by definition of $\delta$-similarity). However, given that we have executed the PRQ-Learning algorithm with a confidence factor of $f$, the obtained gain $W_\Omega$ only satisfies that $W_\Omega \geq fW_{\max}$ by definition of proper execution of the PRQ-Learning algorithm. Thus, $W_\Omega \geq fW_{\max}$, and $W_{\max} > \delta W_\Omega^*$, so $W_\Omega > f\delta W_\Omega^*$. Previous theorems demonstrate theoretically that the basis of a domain can be constructed under several constrains. The next subsection demonstrates empirically that the Basis-Library of the domain can be learned.
6.3 Experiments

This section describes the experiments performed to learn a Basis-Library in the navigation domain presented in Section 4. In this case, performing a task consists on 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 executed. Notice that there is no separation between learning and test, so the correct balance between exploration and exploitation must be achieved to maximize the average gain in each performance.

In the following experiments, 50 different tasks are sequentially performed, each of them with a different reward function, located in different positions of the different rooms of the domain, as shown in Figure 8(a). Notice that 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, the same parameter setting as in section 5.3 is used.

![Figure 8: 50 different goal areas.](image)

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 eigen-policies stored in the Policy Library, shown in Figure 9. The figure 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 eigen-policies is around 14. Interestingly, this is very close to the number of rooms in the domain (15). While increasing $\delta$, the number of eigen-policies increases and when $\delta = 1$, almost all the learned policies are stored.

Figure 10 shows an example of the eigen-policies obtained in one execution, with $\delta = 0.25$. The figure represents the Policy Library obtained after performing the 50 tasks which, as defined above, is composed of 14 eigen-policies. In the figure, we assume that a policy is represented by the goal area of the task that it solves. An eigen-policy is represented also by the goal area, but in this case, the area is shaded. The figure demonstrates that for most of the rooms, one and only one eigen-policy has been learned. 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. That allows
Figure 9: Number of eigen-policies obtained.

us to say that the structure of the domain has been learned by the PLPR algorithm, and is represented by the eigen-policies.

Figure 10: Eigen-Policies ($\delta = 0.25$).

Figure 11(a) shows the average gain obtained when performing the 50 different tasks with the PLPR algorithm, for the different values of $\delta$. 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, as introduced above, $\delta = 0$ generates a Policy Library with only one policy (the first one learned). For comparisons, the same learning process has been executed with different exploration strategies that learn from scratch, and summarized in Figure 11(b).

The average gain obtained while new policies are learned stabilizes around 0.12 for all the strategies, without very significant differences. This demonstrates that Policy Reuse can obtain an increment of almost a 100% gain in the performance of the 50 tasks over the results obtained when the 50 tasks are learned from scratch. Interestingly, when $\delta = 0$, and only one policy is stored, it also obtains improved results over learning from scratch, due to a good behavior of the $\pi$-reuse exploration strategy. That confirms that providing the learning process with a bias improves the performance, even when that bias may not be the best for all the learning processes.
7. Conclusions

Policy Reuse contributes to Reinforcement Learning with three main capabilities. Firstly, it provides Reinforcement Learning algorithms with a mechanism to bias an exploration process by reusing a Policy Library. Our proposed Policy Reuse algorithm, called PRQ-learning, improves the learning performance over some exploration strategies that learn from scratch. Second, Policy Reuse provides an incremental method to build the Policy Library. The library is built at the same time that 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 $\delta$-eigen-policies or $\delta$-Basis-Library. Reusing this set of policies ensures that a minimum gain will be obtained when learning a new task, as demonstrated theoretically.

This work is based on the reuse of past policies to solve similar tasks in the same domain, what can be called intra-domain transfer learning. However, we would like to extend our Policy Reuse method to reuse across different representational frameworks, including different agents or domains. We are interested in understanding how state and action policies are shared and reused across learners with different views of the domain and rewards. Such transfer will allow the action policies of an agent to be transferred to a new teammate performing a similar task. Our ultimate goal is to effectively generalize policies so that they apply to abstracted situations, agents, and domains.

Acknowledgments

This work has been partially funded by grants from the University Carlos III of Madrid, the Spanish Ministry of Education and the Fulbright Commission.

References


