Reusing and Building a Policy Library

Fernando Fernández and Manuela Veloso
Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
15213 Pittsburgh, PA, USA
fernando@cs.cmu.edu, veloso@cs.cmu.edu

Abstract

Policy Reuse is a method to improve reinforcement learning with the ability to solve multiple tasks by building upon past problem solving experience, as accumulated in a Policy Library. Given a new task, a Policy Reuse learner uses the past policies in the library as a probabilistic bias in its new learning process. We present how the effectiveness of each reuse episode is indicative of the novelty of the new task with respect to the previously solved ones in the policy library. In the paper we review Policy Reuse, and we introduce theoretical results that demonstrate that: (i) a Policy Library can be selectively and incrementally built while learning different problems; (ii) the Policy Library can be understood as a *basis* of the domain that represents its structure through a set of core policies; and (iii) given the basis of a domain, we can define a lower bound for its reuse gain.

Introduction

Reinforcement learning is a well known technique for control learning in non-deterministic domains where the state transitions and the reward functions are not necessarily known. Reinforcement learning returns a policy as a function that maps states of the world into the actions that maximize a reward function. A policy is viewed as a universal plan that specifies the actions to be taken by the plan executor at any state. Inspired by our previous work on planning by analogy (Veloso 1994), we investigate how to extend reinforcement learning to reuse past learned policies. The result of our investigation is a successful policy reuse algorithm (Fernández & Veloso 2006). In this paper, we present Policy Reuse, and focus on introducing the reuse and accumulation of a Policy Library.

Policy Reuse uses past learned policies from a policy library as a *probabilistic bias*, in which 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 present the $\pi - reuse$ exploration strategy that defines the balance among these choices. From this strategy, we derive a metric to estimate the similarity of past policies with respect to a new problem. The PRQ-Learning algorithm (Policy Reuse in Q-Learning), is able both to bias a new learning process with policies from a Policy Library, and to use the similarity metric to decide which policies are more effective to reuse (Fernández & Veloso 2006).

The PRQ-Learning algorithm identifies classes of similar policies revealing a *basis*, (the basis library) of *core policies* of the domain. The PLPR algorithm (Policy Library through Policy Reuse), creates such a library of policies. In a nutshell, when solving a new problem through reuse, the PLPR algorithm determines whether the learned policy is or is not "sufficiently" different from the past policies, as a function of the effectiveness of the reuse. The idea is to identify the core policies that need to be saved to solve any new task in the domain within a threshold of similarity. Given a threshold δ defining the performance improvement of reuse, the PLPR algorithm identifies a set of "δ-core policies," as the basis of the domain. The policies represent the core structure of the domain. We provide a formal definition of δ-basis library and δ-core policy. We also describe the conditions that a Policy Library must satisfy so it can be considered a Basis of the domain.

Policy Reuse in Reinforcement Learning

A Markov Decision Process (MDP) is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T}$ is an unknown stochastic state transition function, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re$, 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 problems or *tasks* can be solved, and where we want to transfer knowledge acquired while learning some tasks to new ones. In these cases, the MDP's formalism is not expressive enough to represent all the concepts involved in the knowledge transfer (Sherstov & Stone 2005), so we define domain and task separately to handle different tasks executed in the same domain. We characterize a domain, $\mathcal{D}$, as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T} >$, i.e., the state and action spaces and the transition function. We introduce a task as a specific reward function in a domain, so $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{T}$ stay constant for all the tasks in the same domain.

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

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

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

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

where γ ($0 \leq \gamma \leq 1$) reduces the importance of future rewards, and $r_{k,h}$ defines the immediate reward obtained in the step h of the episode k, in a total of K episodes. Action policies are represented using the action-value function, $Q^\Pi(s, a)$, which defines for each state $s \in \mathcal{S}$, $a \in \mathcal{A}$, the expected reward that will be obtained if the agent starts to act from s, executing a, and after it follows the policy Π. So, the RL problem is mapped to learning the function $Q^\Pi(s, a)$ that maximizes the expected gain.

The 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. The *Policy Library* is the set of past policies.

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

Policy Reuse can be summarized as follows: we want to solve the task Ω, i.e., learn Π^*_Ω; we have previously solved the set of tasks $\{\Omega_1, \ldots, \Omega_n\}$ with n policies stored as a Policy Library, $L = \{\Pi_1, \ldots, \Pi_n\}$; how can we use the policy library, L, to learn the new policy, Π^*_Ω?

We answer this question, by introducing (i) a exploration strategy able to bias the exploration process towards the policies of the Policy Library; (ii) a method to estimate the utility of reusing each of them and to decide whether to reuse them or not; and (iii) an efficient method to construct the Policy Library. We briefly describe these three elements.

The π-reuse Exploration Strategy

The π-reuse strategy biases a new learning process with a past policy. Let Π_{past} be the past policy to reuse and Π_{new} the new policy to be learned. We use a direct RL method to learn the action policy, so we learn the related Q function by applying the Q-Learning algorithm (Watkins 1989). The goal of π-reuse is to balance random exploration, exploitation of the past policy, and exploitation of the new policy, as represented in Equation 2.

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

The equation shows that the π-reuse strategy follows the past policy with a probability of ψ, while it exploits the new policy with a probability of $1 - \psi$. Clearly, random exploration is always required, so when exploiting the new policy, it follows an ϵ-greedy strategy.

Interestingly, the π-reuse strategy also contributes a similarity metric between policies, based on the gain obtained when reusing each policy. Let W_i be the gain obtained while executing the π-reuse exploration strategy, reusing the past policy Π_i. W_i represents an estimation of the similarity between the policy Π_i and the new policy being learned. The set of W_i values, for $i = 1, \ldots, n$, is unknown a priori, but it can be estimated on-line while the new policy is computed in the different episodes. Our PRQ-Learning algorithm formalizes this idea (Fernández & Veloso 2006).

PRQ-Learning Algorithm

Table 1 describes the PRQ-Learning algorithm (Policy Reuse in Q-Learning) (Fernández & Veloso 2006). The learning algorithm used is Q-Learning (Watkins 1989). The goal is to solve a task Ω, i.e., to learn an action policy Π_Ω. We assume that we have a Policy Library $L = \{\Pi_1, \ldots, \Pi_n\}$ composed of n past policies. Then, W_i is the expected average reward that is received when reusing the policy Π_i with the π-reuse exploration strategy, and W_Ω is the average reward that is received when following the policy Π_Ω greedily. The algorithm uses the W values in a softmax manner to choose between reusing a past policy with the π-reuse exploration strategy, or following the ongoing learned policy greedily.

PRQL(Ω, L, K, H)

Given:
- A new task Ω we want to solve
- A Policy Library $L = \{\Pi_1, \ldots, \Pi_n\}$
- A maximum number of episodes to execute, K
- A maximum number of steps per episode, H

Initialize:
- $Q_\Omega(s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
- $W_\Omega = W_i = 0$, for $i = 1, \ldots, n$

For $k = 1$ to K do
- Choose an action policy, Π_k, assigning to each policy the probability of being selected computed by the following equation: $P(\Pi_j) = \frac{e^{\tau W_j}}{\sum_{p=0}^{n} e^{\tau W_p}}$ where W_0 is set to W_Ω
- Execute a Q-Learning episode k:
  - If $\Pi_k = \Pi_\Omega$, exploit greedily the policy Π_k
  - Otherwise, reuse Π_k through the π-reuse strategy
  - In any case, receive the reward obtained in that episode, say R, and the updated Q function, $Q_\Omega(s, a)$
- Recompute W_k using R

Return the policy derived from $Q_\Omega(s, a)$

Table 1: PRQ-Learning.

The PLPR Algorithm

The $PLPR$ algorithm (Policy Library through Policy Reuse) is able to build a library of policies. Initially the Policy Library is empty, $PL = \emptyset$. When the first task Ω_1 is solved, the corresponding learned policy Π_1 can only be learned without reuse. Π_1 is added to the Policy Library and $PL = \{\Pi_1\}$. Upon a second task, the PRQ-Learning algorithm is applied reusing Π_1. Π_2 is learned. The algorithm makes a decision on whether to add Π_2 to the Policy Library, based on how similar Π_1 is to Π_2, following a similarity function based on the efficiency of the reuse episode,

as we introduce below. A similar process continues as multiple tasks are presented to the learner.

Definition 4. *Given a policy, Π_i that solves a task $\Omega_i =< \mathcal{D}, R_i >$, a new task $\Omega =< \mathcal{D}, R_\Omega >$, and its respective optimal policy, Π, Π is δ-similar to Π_i (for $0 \leq \delta \leq 1$) if $W_i > \delta W^*_\Omega$, where W_i is the Reuse Gain of Π_i on task Ω and W^*_Ω is the average gain obtained in Ω when an optimal policy is followed.*

From this definition, we can also formalize the concept of δ-similarity with respect to a Policy Library, L.

Definition 5. *Given a Policy Library, $L = \{\Pi_1, \ldots, \Pi_n\}$ in a domain $\mathcal{D}$, a new task $\Omega =< \mathcal{D}, R_\Omega >$, and its respective optimal policy, Π, Π is δ-similar with respect to L iff $\exists \Pi_i$ such as Π is δ-similar to Π_i, for $i = 1, \ldots, n$.*

Table 2 describes the PLPR algorithm, which is executed each time that a new task needs to be solved. It gets as an input the Policy Library and the new task to solve, and it outputs the learned policy and the updated Policy Library. The Policy Library update equation requires the computation of the most similar policy, which is the policy Π_j such as $j = \arg_i \max W_i$, for $i = 1, \ldots, n$. The gain that will be 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 δ times the gain obtained by using the new policy (W_Ω), where $\delta \in [0, 1]$ defines the similarity threshold, i.e., whether the new policy is δ-similar with respect to the Policy Library.

PLPR(Ω, L, δ)

Given:
- A Policy Library, L, composed of n policies, $\{\Pi_1, \ldots, \Pi_n\}$
- A new task Ω we want to solve
- A δ parameter

Execute the PRQ-Learning algorithm, using L as the set of past policies.
- Receive from this execution Π_Ω, W_Ω and W_{max}, where:
  - Π_Ω is the learned policy
  - W_Ω is the average gain obtained when the policy Π_Ω was followed
  - $W_{max} = \max W_i$, for $i = 1, \ldots, n$
- Update PL using the following equation:

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

Table 2: PLPR Algorithm.

If δ receives a value of 0, the Policy Library stores only the first policy learned, while if $\delta = 1$, mostly all of the policies learned are inserted, due to the fact that $W_{max} < W_\Omega$, given that W_Ω is maximum if the optimal policy has been learned. Different values in the range $(0, 1)$ generate libraries of different sizes. Therefore δ defines the resolution of the library. An extended description of the previous algorithms, and empirical results of their performance in a robot navigation domain can be found in (Fernández & Veloso 2006). The next section contributes new theoretical results.

Theoretical Analysis of Policy Reuse

The PLPR algorithm has an interesting "side-effect," namely the learning of the *structure* of the domain. As the Policy Library is initialially empty, and a new policy is included only if it is different enough with respect to the previously stored ones, depending on the threshold δ, when the policies stored are *sufficiently 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 *core policies* of such domain. In the following, we introduce the formalization of these concepts.

Definition 6. *A Policy Library, $L = \{\Pi_1, \ldots, \Pi_n\}$ in a domain $\mathcal{D}$ is a δ-Basis-Library of the domain $\mathcal{D}$ iff: (i) $\nexists \Pi_i \in L$, such as Π_i is δ-similar with respect to $L - \Pi_i$; and (ii) every policy Π in the space of all the possible policies in $\mathcal{D}$ is δ-similar with respect to L.*

Definition 7. *Given a δ-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 a δ-Core Policy of the domain $\mathcal{D}$ in L.*

The proper computation of the Reuse Gain for each past policy in the PRQ-Learning algorithm plays an important role, since it allows the algorithm 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 will not be either. Thus, we introduce a new concept that measures how accurate the estimation of the reuse gain is.

Definition 8. *Given a Policy Library, $L = \{\Pi_1, \ldots, \Pi_n\}$ in a domain $\mathcal{D}$, and a new task $\Omega =< \mathcal{D}, R_\Omega >$, let us assume that the PRQ-Learning algorithm is executed, and it outputs the new policy Π_Ω, 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 η ($0 \leq \eta \leq 1$), if Π_Ω is optimal to solve the task Ω, and the error in the estimation of both parameters is lower than a factor of η, i.e.,:*

$$\begin{array}{c} \hat{W}_{max} \geq \eta W_{max} \text{ and } \eta \hat{W}_{max} \leq W_{max} \text{ and} \\ \hat{W}_{\Pi_\Omega} \geq \eta W_{\Pi_\Omega} \text{ and } \eta \hat{W}_{\Pi_\Omega} \leq W_{\Pi_\Omega} \end{array} \quad (3)$$

where W_{max} is the actual value of the Reuse Gain of the most similar policy and W_{Π_Ω} 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.

The previous definition allows us to enumerate the conditions that make the PLPR algorithm build a δ-Basis-Library, as described in the following theorem.

Theorem 1. *The PLPR algorithm builds a δ-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 δ-Basis-Library can be demonstrated by induction. The base case

is when the library is composed of only one policy, given that no policy is δ-similar with respect to an empty library. The inductive hypothesis states that a Policy Library $L_n = \{\Pi_1, \ldots, \Pi_n\}$ is a δ-Basis-Library. Lastly, the inductive step is that the library $L_{n+1} = \{\Pi_1, \ldots, \Pi_n, \Pi_{n+1}\}$ is also a δ-Basis-Library. If the PLPR algorithm has been followed to insert Π_{n+1} in L, we ensure that Π_{n+1} is not δ-similar with respect to L, given this is the condition to insert a new policy in the library, as described in the PLPR algorithm. Furthermore, Π_i (for $i = 1, \ldots, n$) is not δ-similar with respect to $L_{n+1} - \Pi_i$, given that (i) Π_i is not δ-similar with respect to $L_n - \Pi_i$ (for inductive hypothesis); and (ii) Π_i is not δ-similar to Π_{n+1} because the reuse distance is symmetric (by second condition of the theorem), and that ensures that if Π_i is not δ-similar to Π_{n+1}, then Π_{n+1} is not δ-similar to Π_i. Lastly, the second condition of the definition of δ-Core 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. The symmetry of 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 the previous theorem requires 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 η, as the following theorem claims.

Theorem 2. *The PLPR algorithm builds a $(2\eta\delta)$-Basis-Library if (i) the PRQ-Learning algorithm is Properly Executed with a confidence of η; (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 Π_{n+1} is inserted in the δ-Core Policy $L_n = \{\Pi_1, \ldots, \Pi_n\}$ following the PLPR algorithm. The policy is added only if it is not δ-similar with respect to L_n. In that case, if the PRQ-Learning algorithm has been properly executed with a confidence of η, we can only ensure that the policy Π_{n+1} is not $(2\eta\delta)$-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 δ-Basis-Library to solve a new task.

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

$$W_\Omega > \eta\delta W^*_\Omega \tag{4}$$

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 Π_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 generated only by the most similar policy, Π_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 δ-Basis-Library we know that every policy Π in the domain $\mathcal{D}$ is not δ-similar with respect to L. Thus, the most similar policy in L, Π_i is such that its Reuse Gain, W_{max} satisfies $W_{max} > \delta W^*_\Omega$ (by definition of δ-similarity). However, given that we have executed the PRQ-Learning algorithm with a confidence factor of η, the obtained gain W_Ω only satisfies that $W_\Omega \geq \eta W_{max}$ by definition of proper execution of the PRQ-Learning algorithm. Thus, $W_\Omega \geq \eta W_{max}$, and $W_{max} > \delta W^*_\Omega$, so $W_\Omega > \eta\delta W^*_\Omega$.

Conclusions

Policy Reuse contributes three main capabilities to control learning. First, it provides Reinforcement Learning algorithms with a mechanism to probabilistically bias a learning process by reusing a Policy Library. Second, Policy Reuse provides an incremental method to build the Policy Library. And third, our method to build the Policy Library allows the learning of the structure of the domain in terms of a set of δ-core policies or δ-Basis-Library. We have defined the conditions required in order for the PLPR algorithm to build the δ-Basis-Library. The reuse of this basis or set of policies ensures that a minimum gain will be obtained when learning a new task, as demonstrated theoretically.

Acknowledgments

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

References

Fernández, F., and Veloso, M. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*.

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

Veloso, M. M. 1994. *Planning and Learning by Analogical Reasoning*. Springer Verlag.

Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, King's College, Cambridge, UK.