Probabilistic Policy Reuse in a Reinforcement Learning Agent
Fernando Fernandez            Manuela Veloso
Carnegie Mellon University
&

Using Spatial Hints to Improve Policy Reuse in a Reinforcement Learning Agent
Bruno N. da Silva            Alan Mackworth
Computer Science Department, University of British Columbia

Presented by: Olimpiya Saha
A Reinforcement Learning Agent-What is it?-A quick glance

• An agent is placed in an unknown environment where it must learn to behave successfully.

• Examples: Chess or any new game.

• Uses the concept of rewards like in MDP to learn the optimal policy.

• Optimal Policy: Aims to maximize the expected total reward.

• No prior knowledge of environment and reward function.
Introduction

• **Policy Reuse**- Introduced as a technique for reinforcement learning guided by past policies.

• Basic aim is to achieve balance between 3 tasks:

  1. Exploitation of the ongoing learned policy.

  2. Adopting randomness in action.

  3. Exploration of past policies.

• Solves the **Exploration Vs Exploitation** tradeoff.

• $\epsilon$-greedy, Boltzmann or Directed Exploration use knowledge obtained in the current learning process.
Outline

• 3 algorithms proposed in this paper:

I. PRQ-Learning Algorithm - Main algorithm.
II. Π-reuse Algorithm
III. PLPR Algorithm

• PRQ - Learning introduces Π-reuse exploration strategy and a similarity function.
• Π-reuse- Probabilistically biases exploration to include given past policy.
• PLPR - Incremental method to build a library of policies.
Definitions

- Reinforcement Learning problems formalized using Markov Decision Processes (MDPs).

- **Definition 1.** An MDP is a tuple \( < S, A, T, R > \), where \( S \) is the set of states, \( A \) is the set of actions, \( T \) is a stochastic state transition function, \( T : S \times A \times S \rightarrow |R| \), and \( R \) is a stochastic reward function, \( R : S \times A \rightarrow |R| \). RL assumes that \( T \) and \( R \) are unknown.

- **Definition 2.** A Domain \( D \) is a tuple \( < S, A, T > \), where \( S \) is the set of all states; \( A \) is the set of all actions; and \( T \) is a state transition function, \( T : S \times A \times S \rightarrow |R| \).

- **Definition 3.** A task \( \Omega \) is a tuple \( < D, R > \), where \( D \) is a domain; and \( R \) is the reward function, \( R : S \times A \rightarrow |R| \).
• **Definition 4.** A Policy Library, $L$, is a set of $n$ policies
   \{1, \ldots, n\}. Each policy $\pi_i \in L$ solves a task $\Omega_i = \langle D, R_{\Omega_i} \rangle$, i.e., each
   policy solves a task in the same domain.

• **Definition 5.** A trial or episode starts by locating the learning agent in a
   random position in the environment. Each episode finishes when the agent
   reaches a goal state or when it executes a maximum number of steps, $H$.

• The agent’s goal is to maximize the expected average reinforcement per
   episode, $W$ which is given by the following equation.

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

   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.
• **Definition 6.** An action policy, $\pi$, is a function $\pi: S \rightarrow A$ that defines how the agent behaves. If the action policy was created to solve a defined task, $\Omega$, we call that action policy $\pi_\Omega$. The gain, or average expected reward, received when executing an action policy $\pi$ in the task $\Omega$ is called $W_\Omega^\pi$.

• **Definition 7.** An action policy $\pi_\omega^*$ is optimal if $W_{\Omega}^{\pi_\omega^*} \geq W_{\Omega}^\pi$ for all policy $\pi$ in the space of all possible policies when $K \rightarrow \infty$.

• **Assumption:** The episodic tasks are solved with absorbing goal states.
$\pi$-reuse Algorithm

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

- Follows the past policy with probability $\psi$, and it exploits the new policy with probability of $1 - \psi$.
- $\epsilon$-greedy strategy used by the new policy.
- Q-Learning integrated with policy reuse.
- Assumed values: $K=2000$, $H=100$, $\gamma=0.95$, $\alpha=0.05$

Table 1: $\pi$-reuse Exploration Strategy.
• A grid-based robot navigational domain with multiple rooms is used for experimental purpose.

• The environment is represented by walls, free positions and goal areas all of which are of size 1x1.

Figure 1: Grid-based Office Domain
- Learning process first executed following different exploration strategy that do not use any past policy.

- Then learning is executed with policy reuse.
The goal of PRQ-Learning is to solve a new task \( \Omega \) and learn an action policy \( \pi_\Omega \).

It has a library of policies as an input composed of \( n \) optimal policies.

The probability of a past policy being selected is given by the expression:

\[
P(\pi_j) = \frac{e^{TW_j}}{\sum_{p=0}^{n} e^{TW_p}}
\]

If the policy chosen is the ongoing learned policy, purely greedy strategy used.

Otherwise, \( \pi \)-reuse technique used.
Experiments and Results

- Experiments are performed with PRQ-Learning Algorithm with 3 different libraries as inputs.

- \[ L_1 = \{\pi_2, \pi_3, \pi_4\} \]
- \[ L_2 = \{\pi_1, \pi_2, \pi_3, \pi_4\} \]
- \[ L_3 = \{\pi_1, \pi_2, \pi_3, \pi_4\} \]

- Assumed values: \[ \tau = 0 \]
  \[ \Delta \tau = 0.05 \]
PLPR Algorithm

- Used for building a library of policies.

- A similarity threshold $\delta$ is considered whose value is between 0 and 1.

- **Definition.** Given a Policy Library, $L = \{\pi_1, \ldots, \pi_n\}$ in a domain $D$, a new task $\Omega = <D, R_\Omega>$, 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 $i = 1, \ldots, n$. 

Conclusions

• Introduced algorithms which address the main challenges of policy reuse in a reinforcement learning agent.
• First, the PRQ-Learning algorithm allows to probabilistically bias an exploration learning process by reusing a Policy Library.
• Improves the learning performance over exploration strategies that learn from scratch.
• The PLPR algorithm incrementally builds the Policy Library.
• Future scope: Extension of the algorithm to include different agents or domains.
Using Spatial Hints to Improve Policy Reuse in a Reinforcement Learning Agent

Bruno N. da Silva       Alan Mackworth
Computer Science Department
University of British Columbia
Introduction

- **Reinforcement Learning** - Popular technique for intelligent agent design.

- **Problem** - Large state and action space.

- **Degradation of performance in a learning agent.**

- **Introduces the idea of Spatial Hints.**

- **Spatial Hints** - Simple information extracted from experts which estimates the portion of the world state in which the existing policies would be useful.
Problem Definitions

• Reinforcement Learning problem defined using a Markov decision process.

• **Definition 1.** A hint $h$ is a pair $<\pi_h, s_h>$ where $\pi_h$ is a policy defined on the domain $D$ and $s_h$ belongs to $S(D)$ is a state in the state space $S$ of $D$. We call $s_h$ the reference point of the policy $\pi_h$.

• **Definition 2.** A hint library $L$ is a set of $n$ hints $\{h_1, \ldots, h_n\}$. $\exists D$ such that each hint $h_i$ in the library $L$ solves a task $\Omega = <D, R_o>$.

• **Definition 3.** A step $t$ begins in a certain state $s_t$ and ends when the agent executes an action $a_t$ and receives a reward $r_t$ for that action in that state. A slot $\sigma$ is a sequence of $k$ steps.

• **Definition 4.** An episode is a sequence of slots $\{\sigma_0, \ldots, \sigma_{k-1}\}$, each of them containing the same number of steps. An episode ends after reaching the maximum number of slots or after the goal state is reached.
• Alternatively, we define an episode as containing k slots. Each slot is occupied by a policy (or an exploration strategy such as \( \epsilon\)-greedy) that will dictate the actions to be taken in the steps of this slot.

• **Definition 5.** An evaluation metric is defined as the average reward per episode.

• In the equation below \( E \) is the number of episodes, \( t_k \) is the number of slots in the episode \( e \) and \( k_e \) is the number of slots in slot \( k \). \( \gamma \) is a value between \([0,1]\) which is the discount factor of future rewards and \( r_{k,t,e} \) is the reward received in step \( t \) of slot \( k \) of episode \( e \).

\[
W(E) = \frac{1}{E} \sum_{e=0}^{E-1} \sum_{k=0}^{k_e-1} \sum_{t=0}^{t_k-1} \gamma^{k+1} r_{k,t,e}
\]
• Hint associates a policy with a single state.
• So some metric needed to estimate how useful each hint is in each state of the world.
• Metric based on the distance between the current state of the world and the policy’s reference point.
• Closer hints more suitable, farther are less.
• Quality of the hint may be non-uniform.
• A quantity \( \text{reach} \) associated with each hint.
• Reach high for better performance around reference and vice versa.
• A metric constructed which is proportional to reach and inversely proportional to the distance between the current and reference states.
• Policies assigned to slots with probability proportional to \( w_i \).

\[
W_i = \frac{\text{reach}_i}{1 + \text{distance}(\text{current state}, \text{reference state}_i)}
\]
• After assignment of a policy $\pi_i$ to a slot, the execution of this slot starts.

• The selection of actions determined by two sources: the existing policy $\pi_i$ and an $\varepsilon$ – greedy procedure based on the Q-values of the current task.

$$\varepsilon-\text{greedy}(\pi_{new})=\text{best action with probability } \varepsilon$$

$$=\text{random action with probability } (1-\varepsilon).$$

Figure 2. Action selection process in a slot combining an existing policy $\pi_i$ and an $\varepsilon$ – greedy procedure.
Algorithms

Table 1 describes the algorithm executed during each slot.

After the policy for that slot is determined, it is passed together with the initial state of the slot to the algorithm.

Iteration repeated for maximum number of states or goal reached.

The second algorithm consists of a sequence of calls to the slot algorithm.

The policy for each slot is selected according to the value of $w_i$. 

<table>
<thead>
<tr>
<th>Algorithm slot($\pi_i$, $s_{initial}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p := 1.00$</td>
</tr>
<tr>
<td>$s_{curr} := s_{initial}$</td>
</tr>
<tr>
<td>$\epsilon := \epsilon_0$</td>
</tr>
<tr>
<td>Repeat T times or until $s_{curr} = s_{goal}$</td>
</tr>
</tbody>
</table>
| action := \begin{cases}   
  \pi_i(s_{curr}), with probability p \\  
  \epsilon - \text{greedy}(\pi_{new}(s_{curr})), with prob 1 - p  
\end{cases}  |
| Execute action, collecting $s_{next}$ and reward $r$ |
| $Q_{\pi_{new}}(s_{curr}, action) := (1 - \alpha) Q_{\pi_{new}}(s_{curr}, action) + \alpha [ r + \gamma \max_a Q_{\pi_{new}}(s_{next}, a) ]$ |
| $p := p - \frac{1}{T}$            |
| $s_{curr} := s_{next}$            |
| $\epsilon := \min (1, \epsilon + \Delta\epsilon)$ |

Table 1. Slot exploration strategy

<table>
<thead>
<tr>
<th>Algorithm episode($s_{initial}, Library$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_{curr} := s_{initial}$</td>
</tr>
<tr>
<td>Repeat K times or until $s_{curr} = s_{initial}$</td>
</tr>
<tr>
<td>Let $w_i = \frac{\text{reach}<em>i}{\text{distance}(s</em>{curr}, i)} \quad \forall i \in Library$</td>
</tr>
<tr>
<td>Select $\pi$ according to distribution $p(\pi_i) = \frac{w_i}{\Sigma_i w_i}$</td>
</tr>
<tr>
<td>Execute slot($\pi$, $s_{curr}$)</td>
</tr>
<tr>
<td>Retrieve new $s_{curr}$ from slot</td>
</tr>
</tbody>
</table>

Table 2. Episode definition strategy
Problem arises when all existing policies are ineffective in some parts of the state space.

Though switching between existing policy and $\epsilon$-greedy is done, this may be inefficient for some states.

Before the start of each episode, an extra entry introduced with $\epsilon$-greedy strategy and an initial state with different reach.

Identifies which value of reach in initial point will give higher rewards for existing policies.
SHPR-Spatial Hints for Policy Reuse Algorithm.

- Repeat a number of times (E-input).
- Next repeat based on the question from the abstract definition (REPETITIONS_PER_EPISODE another input).
- 2 points where reach table is updated.
- 1st immediately after episode and the 2nd after the completion of the experiment.

---

**Algorithm SHPR**

```cpp
For each \( \pi_i \in Library \)

\[
reach_i := INITIAL\_REACH
\]

Repeat E times

\[
s_{\text{initial}} := \text{selectInitialState}()
reach_{\varepsilon\text{-greedy}} := 1
maxReach := \max_{\pi_i \in Library} reach_i
\]

\[
\Delta reach := \frac{\maxReach - reach_{\varepsilon\text{-greedy}}}{\text{REPETITIONS\_PER\_EPISODE}}
\]

\[
\text{accRewards} := 0
\]

\[
\text{accWReach} := 0
\]

While \( reach_{\varepsilon\text{-greedy}} \leq maxReach \)

\[
\text{tempLibrary} := Library \cup \{ \varepsilon\text{-greedy}(\pi_{\text{new}}) \}
\]

with reference at \( s_{\text{initial}}, reach = \text{reach}_{\varepsilon\text{-greedy}} \)

Execute episode \((s_{\text{initial}}, \text{tempLibrary})\)

Retrieve total discounted reward \( R \) from episode

For each \( \pi_i \in Library \)

\[
reach_i := reach_i + participation_i \cdot R,
\]

where \( participation_i = \frac{t \_ \text{slots from last episode using } \pi_i}{t \_ \text{slots from last episode}} \)

\[
\text{accWReach} := \text{accWReach} + reach_{\varepsilon\text{-greedy}} \cdot R
\]

\[
\text{accRewards} := \text{accRewards} + R
\]

\[
reach_{\varepsilon\text{-greedy}} := reach_{\varepsilon\text{-greedy}} + \Delta reach
\]

\[
reach_{\varepsilon\text{-greedy}} := \frac{\text{accWReach}}{\text{accRewards}}
\]

Library := Library \cup \{ \varepsilon\text{-greedy}(\pi_{\text{new}}) \}

with reference at \( s_{\text{initial}}, reach = \text{reach}_{\varepsilon\text{-greedy}} \)
```

---

**Table 4. The Spatial Hints for Policy Reuse algorithm**
Experiments and Results

- **Purpose**: To evaluate this algorithm against two baselines: PRQL Algorithm and Q-Learning Algorithm.
- The 5 tasks considered for experiments:
  - Red point-Goal
  - Green point-Reference
• **Library** = \( \{\pi_1, \pi_2, \pi_4\} \)

• \(\Omega\) and \(\Omega_4\) are similar to the task for which the policy is to be learned.

• Both PRQL and SHPR perform well.
Figure 5. Comparison using existing policies $\pi_1$, $\pi_2$, $\pi_3$, and $\pi_4$. This library contains entry 3 which should be detrimental to SHPR and PRQL. The legend displays the entries in order of final accumulated reward, from highest to lowest.
Conclusions

• Introduced the concept of Spatial Hints Policy Reuse (SHPR).
• Contributed in showing how simple information elicited from user can lead to considerable efficiency.
• This information extraction requires minimal cost.
• Method robust to unfavourable inputs.
• Extracts information from past policies that share some degree of similarity with current task.