

Reinforcement Learning in the RoboCup-Soccer Keepaway

Javier García¹, Fernando Fernández², and Manuela Veloso³

¹ Universidad Carlos III de Madrid, Avda. de la Universidad 30, Madrid, Spain
fjaviergp@gmail.com,

² Universidad Carlos III de Madrid, Avda. de la Universidad 30, Madrid, Spain
ffernand@inf.uc3m.es,

<http://www.plg.inf.uc3m.es/~ffernand/>

³ Carnegie Mellon University, 5000 Forbes Avenue Pittsburgh, USA
veloso@cs.cmu.edu,
www.cs.cmu.edu/~mmv/

Abstract. Many researchers purpose reinforcement learning (RL) as a form of machine learning for behavior learning. However, there are several issues that need to be considered when applying RL techniques for solving new tasks. There are many different RL algorithms available such as *Q-learning* or *Sarsa*. These algorithms may produce different results. In complex domains with large state and action spaces is necessary to apply generalization techniques such as function approximation. Last, a right balance between exploration and exploitation is required. In this paper we review these issues in order to improve the learning process in the Keepaway domain. We present some new combinations in the choice of the RL algorithm, the generalization method and the exploration-exploitation strategy

1 Introduction

The objective of reinforcement learning is to acquire knowledge to better decide what action an agent should choose at any moment to optimally achieve a goal [6]. The studies indicate that different kinds of learning mechanism may derive different results. In detail, this paper compares simulation results of Q-learning and Sarsa agents in the Keepaway domain [5]. In addition, except in very small environments it's impossible to enumerate the state and action space and store tables of values over them. The problem of learning in large spaces is addressed as *generalization techniques* [6]. In this paper, we review the VQ method and the CMAC techniques applied to the keepaway domain. Finally, we also review some classical exploration strategies, such as ϵ -greedy, but also new transfer learning approaches as Policy Reuse [2]. Policy Reuse balances among exploitation of the ongoing learned policy, exploration of random actions and exploration of policies previously learned following the π -reuse strategy.

Combining the three issues, learning algorithm, generalization method and exploration strategy may produce different results. In this paper, we test several of these combinations, which was never tested before.

This paper is organized as follows. Section 2 describes the techniques that we will use in the experiments. Section 3 describes the models which integrate the generalization techniques with the learning algorithms, $VQ - TD(\lambda)$ and $CMAC - TD(\lambda)$. The model $PR - TD(\lambda)$ is also introduced which implements the policy reuse ideas. Section 4 describes the experiments performed in the keepaway domain. Lastly, in Section 5 we present our conclusions.

2 Approaches to improve RL processes

In this section, different RL algorithms, generalization methods and exploration strategies are summarized.

2.1 Learning Algorithms: *Watkins's Q*(λ) and *Sarsa*(λ)

Among many different (RL) algorithms, *Q-learning* and *Sarsa* have been widely used. The main difference between *Q-learning* and *Sarsa* is how they update the action-value function [6]. On the one hand, *Sarsa* is an on-policy method and it estimates the action-value function for the behavior policy followed, and at the same time changes the policy followed toward greediness with respect to the action-value function (equation 1).

$$Q(s_t, a_t) \rightarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (1)$$

On the other hand, *Q-learning* is an off-policy method where the learned function, Q , directly approximates the optimal action-value function, independent of the policy being followed (equation 2).

$$Q(s_t, a_t) \rightarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (2)$$

The TD methods, such as *Q-Learning* and *Sarsa*, are based on just the one next reward, using the value of the state one step later as a proxy for the remaining rewards. In the Monte Carlo methods, on the other hand, the update function is based on the entire sequence of observed rewards from that state until the end of the episode. One kind of intermediate method, then, would update the action-value function based on an intermediate number of rewards: more than one, but less than all of them until termination. This methods use a mechanism called eligibility traces that use the λ parameter to shift from one-step TD methods, such as *Sarsa* or *Q-learning*, to Monte Carlo methods [6]. Any TD method can be combined with eligibility traces to obtain a more general method. A method proposed that combine eligibility traces and *Q-learning* is *Watkins's Q*(λ) and a method that combine eligibility traces and *Sarsa* is *Sarsa*(λ). We will use *Watkins's Q*(λ) and *Sarsa*(λ) in our tests.

2.2 Space generalization in Reinforcement Learning

To apply the reinforcement learning methods to task with continuous state space, we use two techniques: VQ and CMAC.

Vector Quantization: VQ

Applying VQ techniques we reduce the size of the reinforcement learning table and we find a more compact representation of the state space [1]. A vector quantizer Q of dimension K and a size N is a mapping from a vector (or a "state") in the K -dimensional Euclidean space, R^k , into a finite set C containing N states, $Q: R^k \rightarrow C$, where $C = \{y_1, y_2, \dots, y_N\}$, $y_i \in R^k$.

In this way, given C , and a state $x \in R^k$, $VQ(x)$ assigns x to the closest state from C , $VQ(x) = \arg \min_{y \in C} \{dist(x, y)\}$. To design the vector quantizer we use the Generalized Lloyd Algorithm [4]. The GLA is a clustering technique that consists of a number of iterations, each one recomputing the set of more appropriate partitions of the input states and their centroids.

CMAC

Consider a problem in which the state set is continuous and 2-dimensional. In this case, a state is a vector with two components. One kind of feature are those corresponding to circles in state space. If a state is in a circle, then the corresponding feature has the value 1, otherwise the feature has the value 0 [6]. Figure 1 (a) show an example which the binary feature vector corresponding to the state $S' = (S'_1, S'_2)$ is $\phi(S') = \{0, 1, 0, 0, 1, 0\}$.

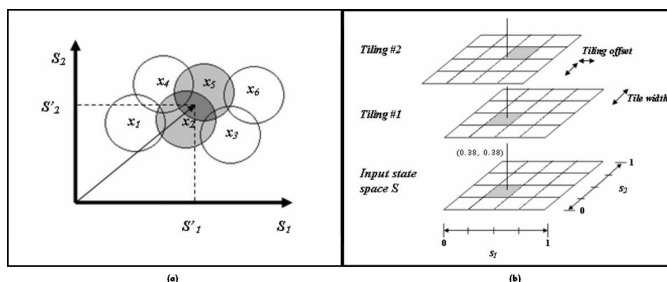


Fig. 1. (a) Coarse coding. (b) Tile coding

CMAC, also known as tile coding, is a form of coarse coding [5, 6]. In CMAC the features are grouped into partitions of input state space. Each of such partition is called a *tiling* and each element of a partition is called a *tile*. Each *tile* is a binary feature. The tilings were overlaid, each offset for the others. In each tiling, the state is in one tile. In the figure 1 (b) we present an example of tile-coding which represent a 2-dimensional input state space S where two tilings were overlaid each offset for the other by $1/2$ of the tile width. The set of all these active tiles, one per tiling and 2 per state, is what makes up the binary feature vector. The binary features vector for the state $(0.38, 0.38)$ is $\phi_s = \{0_0, 0_1, \dots, 1_5, 0_6, \dots, 1_{26}, 0_{27}, \dots, 0_{31}\}$.

The approximate value function, Q_a , is represented not as a table, but as a parameterized form with parameter vector θ_t . This means that the approximate value function Q_a depends totally on θ_t . In CMACs, each tile has associated a

weight. The set of all these weights is what makes up the vector θ . The approximate value function, $Q_a(s)$ is calculated in the equation 3.

$$Q_a(s) = \theta^T \phi = \sum_{i=0}^n \theta(i) \phi(i) \quad (3)$$

2.3 The exploration strategy: $\pi - reuse$

In order to obtain the best reward, an agent must prefer an action that it has tried in the past and found to be effective. The agent has to exploit what it knows in order to obtain the best reward, but it also has to explore in order to make better action choices in the future [6]. For instance, ϵ -greedy selects greedily the best action with probability of ϵ , and selects a random action with probability $(1 - \epsilon)$. However, traditional exploration strategies do not include knowledge acquired in past learning processes. Nevertheless, past policies can be reused to solve new tasks. For instance, $\pi - reuse$ [2] follows the ϵ -greedy strategy with a probability of ψ , but follows a previously learned policy with probability $(1 - \psi)$. Then, $\pi - reuse$ can be used to balance random exploration, exploitation of the past policies and exploitation of the new policy.

3 The models

This section describes the models implemented taking some of the different combinations of algorithms, generalization methods and exploration strategies described in previous section. All of them will be evaluated later in the experimentation section over the Keepaway domain.

3.1 VQ and Reinforcement Learning: $VQ - TD(\lambda)$

$VQ - TD(\lambda)$ is a new approach that extends from the model VQQL [1]. $VQ - TD(\lambda)$ model uses *Watkins's* $Q(\lambda)$ and *Sarsa*(λ) as reinforcement learning algorithms, integrating them with vector quantization techniques. It requires two consecutive phases.

Learn the quantizer. To design the vector quantizer from input data obtained from the environment.

Learn the Q function. Once we have clustered the space state, it is needed to learn the Q function applying *Watkins's* $Q(\lambda)$ or *Sarsa*(λ), generating the Q table, that will be composed of N rows and 1 column for each action.

3.2 CMAC and Reinforcement Learning: $CMAC - TD(\lambda)$

This model uses *Watkins's* $Q(\lambda)$ and *Sarsa*(λ) as reinforcement learning algorithms, combining them with CMAC. In this case, the model doesn't need to design a vector quantizer. It requires two consecutive phases.

Extract features. First, the model extract the features from the state obtained from the environment as described in section 2.2.

Learn the Q function. The model uses these features to learn the Q function (equation 3). Learning is performed by updating the vector θ applying *Watkins's* $Q(\lambda)$ or *Sarsa*(λ).

3.3 Policy Reuse and Reinforcement Learning: $PR - TD(\lambda)$

In this paper we use an algorithm called $PR - TD(\lambda)$ that implements the policy reuse ideas [2, 3]. In complex domains with large state space, $PR - TD(\lambda)$ can use VQ or CMAC to generalize the state space. Finally, it updates the Q function applying *Watkins's* $Q(\lambda)$ or *Sarsa*(λ)

Table 1 summarizes the different models implemented, taking into account the algorithm, the generalization method and the exploration strategy applied.

Model	Algorithm	Generalization Method	Exploration Strategy
VQ-TD(λ)	VQ	$Q(\lambda)$, Sarsa(λ)	ϵ -greedy
CMAC-TD(λ)	CMAC	$Q(\lambda)$, Sarsa(λ)	ϵ -greedy
PR-TD(λ)	VQ, CMAC	$Q(\lambda)$, Sarsa(λ)	ϵ -greedy, π -reuse

Table 1. Summary of the models implemented

4 Experiments

4.1 Keepaway domain

In order to verify the usefulness of these models we have selected a robotic soccer domain, the keepaway domain [5]. In this subtask, the keepers try to maintain the possession of the ball within a region, while the takers try to gain the possession. An episode ends when the takers take the possession or the ball leaves the playing region. When an episode ends, the players are reset for another episode. In our experiments each keeper learns independently. From a point of view of a keeper, an episode consists of a sequence of states, actions and rewards:

$$s_0, a_0, r_1, s_1, \dots, s_i, a_i, r_{i+1} \quad (4)$$

where a_i is selected based on some perception of s_i . We reward the keepers for each time step which they keep possession, so we set the reward r_i to the number of time steps that elapsed while following action a_{i-1} : $r_i = t_i - t_{i-1}$. The keepers' goal is to select an action that the remainder of the episode will be as long as possible, and thus maximize the total reward.

The state space is composed of several features that can be considered continuous. These features use information derived from distances and angles between the keepers, takers and the center of the playing area. The number of features used for each state depends on the number of players. In 3vs2 there are 13 features, in 4vs3 there are 19 and in 5vs4 there are 25 features. In the experiments, we use VQ and CMAC as methods for state space generalization in order to

improve the learning process. The action space is composed by two different macro-actions, $HoldBall()$ and $PassBall(k_i)$ where k_i is the teammate i .

The parameter settings has been performed by a extensive experimentation.

4.2 VQ – TD(λ) and Keepaway

In this case, we use the VQ method, and more specifically, the GLA to discretize the state space. First, we design the vector quatizer composed of $N=64$ centroids from the input data obtained during an random interaction between the keeper and the environment. Once a vector quantizer is designed for each keeper, the Q function must be learned, using a reinforcement learning algorithm such as *Watkins's Q(λ)* or *Sarsa(λ)*, generating the Q table composed of N rows and 1 column for each action.

The parameter setting is the following: $\lambda = 0.2$, $\gamma = 1$, $\alpha = 0.125$. An ϵ – greedy strategy is followed, increasing the value of epsilon from 0 (random behaviour) to 1 (fully greedy behaviour) by 0.0001 in each episode. The size of the playing region is 25×25 . In all the cases, the size of the discretized state space is 64.

In 3vs2, 4vs3 and 5vs4, we obtain the results shown in Figure 2. *Watkins's Q(λ)* and *Sarsa(λ)* have the same parameter setting.

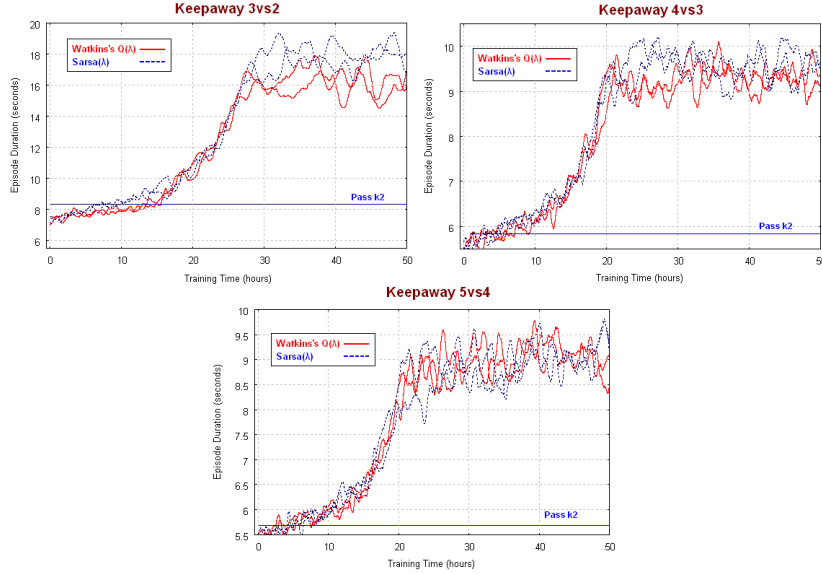


Fig. 2. (a) Experiments in 3vs2, 4vs3 and 5vs4 case.

The y -axis represent the average time that the keepers are able to keep the ball from the takers and the x -axis is training time. In the graphs, there are two learning curves using *Watkins's Q(λ)* and two learning curves using *Sarsa(λ)*,

representing two different executions of each algorithm. From the graphs, we can see that *Watkins's* $Q(\lambda)$ and *Sarsa*(λ) obtain very similar results. In the 3vs2 case, the average values raises from 7 seconds up to around 18 seconds. In the 4vs3 case, the performance raises from random behavior to 10 seconds approximately. Finally, in 5vs4 keepaway, the average value has achieved the 9 seconds.

4.3 CMAC – TD(λ) and Keepaway

In the 3vs2 case, with three keepers against two takers, there are 13 state variables. The intervals of the variables in 3vs2 case in a 25×25 region are defined in table 2.

In our experiments we use single-dimensional tilings. For each variable, 32 tilings were overlaid, each offset from the others by $1/32$ of the tile width. For each state variable, we specified the width of the tiles based of the width of the generalization that we desired. For example, distances were given widths of about 3.0 meters, whereas angles were given widths of about 10.0 degrees [5].

In the 3vs2 case, for state variable x_1 , there are 6 tiles per tiling and each tiling offset from the others by $1/32$ of the tile width. The number of tiles per tiling for each state variable are defined in Table 2.

Feature	Interval	Tile width	Tiles per tiling
dist(K_1, C)	[0, 17.67]	3	6
dist(K_1, K_2)	[0, 35.35]	3	11
dist(K_1, K_3)	[0, 35.35]	3	11
dist(K_1, T_1)	[0, 35.35]	3	11
dist(K_1, T_2)	[0, 35.35]	3	11
dist(K_2, C)	[0, 17.67]	3	6
dist(K_3, C)	[0, 17.67]	3	6
dist(T_1, C)	[0, 17.67]	3	6
dist(T_2, C)	[0, 17.67]	3	6
Min(dist(K_2, T_1), dist(K_2, T_2))	[0, 35.35]	3	11
Min(dist(K_3, T_1), dist(K_3, T_2))	[0, 35.35]	3	11
Min(ang(K_2, K_1, T_1), ang(K_2, K_1, T_2))	[0, 180]	10	18
Min(ang(K_3, K_1, T_1), ang(K_3, K_1, T_2))	[0, 180]	10	18

Table 2. Number of tiles per tiling

Then, the size of the primary vector θ and eligibility traces vector e in 3vs2 is 4224 ($x_{1_{tiles}} + x_{2_{tiles}} + \dots + x_{13_{tiles}}$). In our work, we use a separate value function for each of the discrete actions. So there would be roughly 4224 tiles for the *HoldBall*() action, 4224 tiles for the *PassBall*(k_1) action and 4224 tiles for the *PassBall*(k_2) action (about 12672 total). Thus, there are 12672 tiles in 3vs2 case, 24960 tiles in 4vs3 case and 41280 tiles in 5vs4 case.

The parameter setting is the following: $\lambda = 0.5$, $\gamma = 1$, $\alpha = 0.125$. An ϵ – *greedy* strategy is followed, increasing the value of epsilon from 0 to 1 by 0.0001 in each episode. The size of the playing region is 25×25 .

In the 3vs2, 4vs3 and 5vs4 case we obtain the results shown in Figure 3. The learning curves for these runs use *Watkins's* $Q(\lambda)$ and *Sarsa*(λ) under the same conditions.

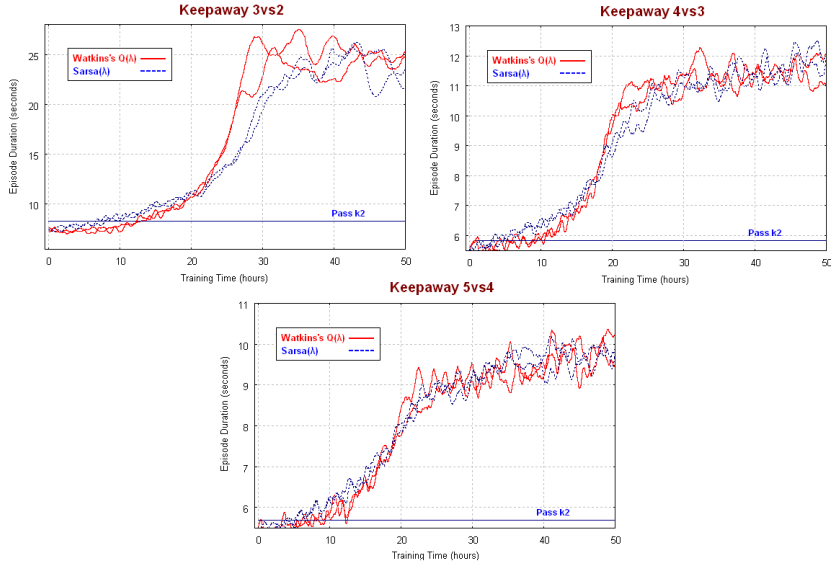


Fig. 3. (a) Experiments in 3vs2, 4vs3 and 5vs4 case.

In the graphs, there are two learning curves using *Watkins's* $Q(\lambda)$ and two learning curves using *Sarsa*(λ). We can see that *Watkins's* $Q(\lambda)$ and *Sarsa*(λ) obtain very similar results, so we can't establish differences. In the 3vs2 case, the average values raises from 7 seconds up to around 25 seconds. In the 4vs3 case, the performance grows from random behavior to 12 seconds approximately. Finally, in 5vs4 keepaway, the average value has achieved the 10 seconds.

4.4 $PR - TD(\lambda)$ and Keepaway

Following the ideas introduced in [3], there are three different keepaway configurations, sequentially learned as defined in table 3.

Next, we performance two types of experiments. First, we VQ to discretize the space state. Later, we apply *Watkins's* $Q(\lambda)$ to learn the Q function. In this case, the parameter setting is the following: In the *Watkins's* $Q(\lambda)$, $\alpha = 0.125$ and $\gamma = 1$.⁴ The size of the discretized state space is 64 states.

In the second set of experiments, we use CMAC to discretize the state space and later we apply *Sarsa*(λ) to learn the Q function. In this case, the parameter

⁴ In the $\pi - reuse$ strategy, $\varphi = 1$, $\nu = 0.95$ and $\epsilon = 1 - \gamma_h$. The parameter τ in the softmax equation is initialized to 0 and incremented by 0.05 in each episode.

	3vs2	4vs3	5vs4
Keeper 1	Learn $\Pi_{3vs2}^{k_1}$	Learn $\Pi_{4vs3}^{k_1}$ by reusing $L^{k_1} = \{\Pi_{3vs2}^{k_1}\}$	Learn $\Pi_{5vs4}^{k_1}$ by reusing $L^{k_1} = \{\Pi_{3vs2}^{k_1}, \Pi_{4vs3}^{k_1}\}$
Keeper 2	Learn $\Pi_{3vs2}^{k_2}$	Learn $\Pi_{4vs3}^{k_2}$ by reusing $L^{k_2} = \{\Pi_{3vs2}^{k_2}\}$	Learn $\Pi_{5vs4}^{k_2}$ by reusing $L^{k_2} = \{\Pi_{3vs2}^{k_2}, \Pi_{4vs3}^{k_2}\}$
Keeper 3	Learn $\Pi_{3vs2}^{k_3}$	Learn $\Pi_{4vs3}^{k_2}$ by reusing $L^{k_3} = \{\Pi_{3vs2}^{k_3}\}$	Learn $\Pi_{5vs4}^{k_2}$ by reusing $L^{k_3} = \{\Pi_{3vs2}^{k_3}, \Pi_{4vs3}^{k_3}\}$
Keeper 4	Not playing	Learn $\Pi_{4vs3}^{k_4}$	Learn $\Pi_{5vs4}^{k_4}$ by reusing $L^{k_4} = \{\Pi_{4vs3}^{k_4}\}$
Keeper 5	Not playing	Not playing	Learn $\Pi_{5vs4}^{k_5}$

Table 3. Description of the tasks.

setting is the following: In the *Sarsa*(λ) algorithm $\lambda = 0.5$, $\gamma = 1$, $\alpha = 0.5$ ⁵ An $\epsilon - greedy$ strategy is followed, increasing the value of epsilon from 0 (random behaviour) to 1 (fully greedy behaviour) by 0.0001 in each episode. The size of the playing region is 25×25 .

The results are shown in the figure 4. The *x-axis* describes the training time, while the *y-axis* shows the episode duration. We compare two different approaches for learning: learning from scratch and Policy Reuse following the method introduced in table 3. In 4vs3, using VQ and *Watkins's* $Q(\lambda)$, when the agents reuse the policy learned, the initial value is 6.2 growing up to 9 seconds. On the other hand, when the agents use CMAC and *Sarsa*(λ), the performance grows from 6 seconds to 11 seconds. In the 5vs4 case, using VQ and *Watkins's* $Q(\lambda)$, when reusing the past policies, the average values raises from 6.5 seconds up to around 9 seconds. However, when the agents use CMAC and *Sarsa*(λ), the performance grows from 6 seconds up to around 9.5 seconds.

5 Conclusions

In the experiments presented in this report, *Watkins's* $Q(\lambda)$ and *Sarsa*(λ) obtain similar results, so we can not establish differences on this domain. In complex domains with a large state space, the choice of the discretization method may have dramatic effect on the results that we obtain. For this reason, the state representation is chosen with great care. In our experiments we have used two discretization techniques: VQ and CMAC. CMAC results are better than VQ results in 3vs2, 4vs3 and 5vs4. Policy Reuse is a method to improve learning performance in new tasks by reusing past policies learned in previous tasks with a different state space and action space. Policy Reuse, combined with generalization state space as VQ or CMAC, is applicable in domains with a large state space. The experiments in the keepaway domain demonstrate that policy reuse improves the behavior of the learning process since the early steps of simulation.

⁵ In the $\pi - reuse$ strategy, $\varphi = 1$, $\nu = 0.95$ and $\epsilon = 1 - \gamma_h$. The parameter τ in the softmax equation is initialized to 0 and incremented by 0.009 in each episode.

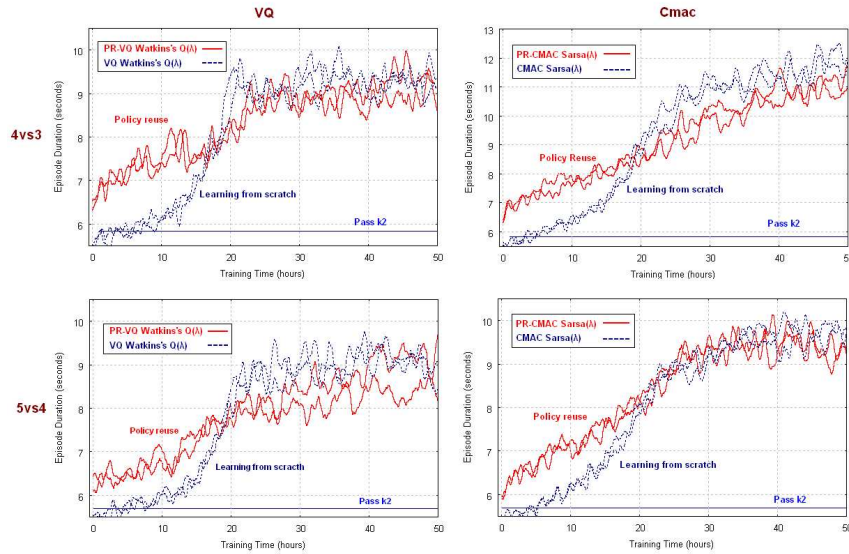


Fig. 4. (a) Experiments PR-*VQ* Watkins's $Q(\lambda)$. (b) Experiments PR-*CMAC* Sarsa(λ)

6 Acknowledgements

This work has been partially supported by the Spanish MEC project TIN2005-08945-C06-05 and regional CAM-UC3M project CCG06-UC3M/TIC-0831.

References

- [1] Fernández, F., Borrajo, D.: VQQL: Applying vector quantization to reinforcement learning. In Robocup-99: Robot soccer world cup III, no.1856 in Lecture Notes in Artificial Intelligence, 292-303. Springer Verlag.
- [2] Fernández, F., Veloso, M.: Probabilistic policy reuse in a reinforcement learning agent. Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'2006). 2006
- [3] Fernández, F., Veloso, M.: Policy Reuse for Transfer Learning Across Tasks with Different State and Action Space. ICML workshop on Structural Knowledge Transfer for Machine Learning. 2006.
- [4] Lloyd, S. P.: Least squares quantization in PCM. Technical Note, Bell Laboratories, 1957.
- [5] Stone, P., Sutton, R., Kuhlmann, G.: Reinforcement Learning for RoboCup-Soccer Keepaway. Adaptive Behaviour 13(3) (2005)
- [6] Sutton, R. S., Barto, A. G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA. 1998.
- [7] Taylor, M., Stone, P.: Behaviour Transfer for Value-Function-Based Reinforcement Learning. Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI'05). 2005.

- [8] Taylor, M., Whiteson, S., Stone, P.: Transfer Learning for Policy Search Methods. International Joint Conference on Autonomous Agents and Multiagent Systems, (AAMAS'07). 2007.