Towards a Continuous Reinforcement Learning Module for Navigation in Video Games

Thierry Gourdin and Olivier Sigaud

LIP6/AnimatLab
8 rue du Capitaine Scott
75015 PARIS
Thierry.Gourdin@lip6.fr, Olivier.Sigaud@lip6.fr

Abstract. Video games are highly non-stationary environments. Our goal is to build a navigation module for video games based on Continuous Reinforcement Learning techniques. A study of the state-of-the-art of these techniques reveals that memory-based approaches are particularly suitable for our application context. More precisely, among memory-based reinforcement learning techniques, we compare a *case-based* approach, proposed by Santamaria, Sutton and Ram to an *instance-based* approach, proposed by Smart and Kaelbling. We show on the standard version of *Mountain-Car* benchmark problem that our modified version of the former converges faster than the latter. Then we show that our algorithm can deal with different non-stationary extensions of the same problem, which is a first step towards the application to video games.

1 Introduction

The video games industry is growing fast. The growth of computational power of personal computers has first been translated into improved visual rendering, resulting in a more realistic immersion of the players into the simulated worlds they play with. Nowadays, more and more game development companies are looking for more realistic behaviors for the Non Player Characters (NPCs, or bots) involved in the games. This results in a surge of interest for Artificial Intelligence (AI) techniques, as exemplified in several recent game development related conferences (GameOn, Game Developers Conference, SIGGRAPH, AAAI and IJCAI workshops).

From the perspective of AI laboratories, the video games industry offers an attractive application domain: the realistic nature of their simulated worlds makes them as interesting as robotic applications, but at a much lower cost, both financially and in terms of experimental effort since the experiments can be run for weeks without the difficulties inherent to robotics.

In particular, from a Reinforcement Learning (RL) perspective, these applications are convenient since it is often easier to define punishments and rewards in the context of a game than to design a suitable behavior for any situation. But video games are also a challenging domain because they are generally multiagent (with human players involved), thus highly non-stationary and unpredictable,

they are generally continuous in nature even if they can be discretized and the perception of NPCs is limited, resulting in partial observability problems.

Recently, Robert [1] has designed a dedicated Learning Classifier Systems (LCSs) architecture combined with a multiobjective action selection mechanism in order to control a team of soldiers in *Team Fortress Classic* (Valve ®). Thanks to RL mechanisms, his agents were able to defeat a medium level team of hand-coded bots called HBPBot and to rival the much more accurate team called FoxBot. One of the reasons of the success of Robert's work is that it relies on a carefully chosen set of high level discrete perceptions and actions. In particular, as far as navigation is concerned, his bots can only choose among a very limited set of destinations and then the classical A^* algorithm [2] is used to define the path from the current location of the bot to its destination.

Instead of the navigation mechanism used by Robert, our long term goal is to use a continuous state and action spaces RL mechanism able to optimize the movement of the bot at any moment given its current context and objectives. This paper presents a preliminary work dedicated to the identification of the most suitable technique in this challenging non-stationary application context.

More precisely, in the next section, we present a state-of-the-art of RL techniques dedicated to continuous state and action spaces, and explain why the so called "memory-based" approaches appear the best choice. There are two classes of "memory-based" algorithms, called "case-based" and "instance-based". In section 3, we present our own model which derives from a case-based algorithm published by Santamaria, Sutton et Ram [3]. In section 4, we use the well-known *Mountain-Car* benchmark problem to experimentally compare our algorithm to the instance-based algorithm called HEDGER from Smart and Kaelbling [4]. We also examine the behavior of our algorithm in the context of a non-stationary version of the *Mountain-Car* problem and discuss the fact that the faster convergence of our algorithm makes it more suitable for the context of non-stationary environments that we will face in video games. In section 5, we conclude to the efficiency of our approach and discuss the extensions that will be necessary to face the more challenging context of commercial video games.

2 Continuous State and Action RL

2.1 Background

The Markov Decision Processes (MDPs) framework [5] is probably the best understood and most suitable mathematical framework when one wants to model the sequential interaction of an agent with its environment, particularly when this interaction is uncertain or stochastic. The framework defines:

- a finite set S of states s and a finite set A of actions a,
- a transition function $T : S \times A \rightarrow \Pi(S)$ which maps (s_t, a_t) couples to probability distributions over the next state s_{t+1} if a_t is performed in the state s_t. Given the probabilistic nature of transitions, $T(s_t, a_t)(s_{t+1})$ is also written $Pr(s_{t+1}|s_t, a_t)$,

– a scalar reward function $R : S \times A \rightarrow \mathbb{R}$ which defines the immediate reinforcement signal that the agent will get if it makes action a in state s.

The Markov property is verified when the probability distribution over the next state can be exactly computed knowing only the current state and the action selected by the agent :

$$Pr(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = Pr(s_{t+1}|s_t, a_t)$$

Mapping an action to each state defines a policy π. The framework was first used to define and solve the so called "planning problem", i.e. find the policy an agent should follow in order to maximize its return, expressed as some function of the rewards received at each time step from the environment [6]. The most common return function is the discounted return: $E^\pi(s_0) = \sum_{t=0}^{\infty} \gamma^t R_t$, where the discount factor γ reflects the relative importance of short or long term rewards and R_t is the reward received at time t.

These methods consist in introducing a *value function* V^π where $V^\pi(s)$ represents the expected return of an agent if it follows policy π from state s. It is shown that, when the Markov hypothesis holds, this function is solution of the Bellman equation [5]:

$$\forall s \in S, V^\pi(s) = \sum_a \pi(s_t, a_t)[R(s_t, a_t) + \gamma \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t)V^\pi(s_{t+1})] \quad (1)$$

From equation 1, the optimal value function V^* can be reached using Dynamic Programming (DP) methods such as Policy Iteration [7, 8] and Value Iteration [6]. Instead of the value function V, it is often more convenient to introduce a function Q where $Q(s, a)$ evaluates the quality of doing action a in state s. Everything that has been said about the function V can be transposed to the function Q, given that $V(s) = \max_a Q(s, a)$.

The problem with DP methods is that they require a perfect knowledge of the transition and reward functions. Such a requirement cannot be generally satisfied in complex and unpredictable environments such as video games. But the same framework can also be used in order to define and solve the so called "learning problem", i.e. reach the optimal policy when the transitions between states and the sources of reward are not known in advance [9].

The counterpart of DP methods in the context of learning problems are called Temporal Difference (TD) methods. The first TD methods whose convergence to optimality was proved are TD, Sarsa and Q-learning [10–12].

TD(0) The basic TD algorithm, called TD(0) [13], is based on a comparison between the reward actually received and the expected reward given the previous estimates. More precisely, the *temporal difference error* $\delta = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ [13] corresponds to the error between the actual values of estimates of $V(s_t)$ and the values they should have. The TD method, whose convergence is

proved in [14], consists in correcting $V(s_t)$ little by little thanks to a Widrow-Hoff equation using a *learning rate* α:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2}$$

However, in a RL context, if the agent does not know the transition function, it cannot derive efficiently a policy from the value function: it does not know which action it should execute to reach the next state with the highest value. This explains why TD(0) is not used in practice when the model of transitions is unknown. Rather than estimating the value function V, most RL algorithms rely on the estimation of the Q-function.

Sarsa The SARSA algorithm is the counterpart of the TD algorithm when one uses the Q-function rather than the value function. Its update rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3}$$

The name SARSA comes from the necessary information for such an update, the quintuplet $(s_t, a_t, R_{t+1}, s_{t+1}, a_{t+1})$. Thus, in order to compute this update, the agent must know in advance its next state s_{t+1} and the action a_{t+1} it is going to take in that state. Such a method is said "On-Policy" since it implies a strong dependency between the policy of the agent and its ability to update its model of the Q-function.

Q-learning Q-LEARNING is simpler than SARSA. Its update rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{4}$$

The term $Q(s_{t+1}, a_{t+1})$ in equation 3 has been replaced by $max_a Q(s_{t+1}, a)$ in equation 4. This time, the update rule is independent of what the agent will do next, thus the algorithm is said "Off-policy". This brings several practical advantages (in particular, the learning process is more independent of the decision process and seems more robust, see [15, 16] for a discussion) but also results in a simpler proof of convergence [11]. Note that both equations would be identical if the agent was following a greedy policy, but it is not the case in general since an agent must explore its environment in order to learn.

Discussion From the background section above, we can already draw two conclusions on the most suitable methods for our application problem:

- DP methods cannot be used as such in the context of video games, because the dynamics of the interaction between agents and their environment is too complex and unpredictable to be modeled by hand. There also exists *model-based* RL methods such as DYNA methods [17] that learn the model of the environment before applying DP methods, but cannot be applied here because learning a non-stationary model of transition and rewards is too difficult so far. Thus we have to rely on plain TD methods;

– among TD methods, we prefer the Off-policy methods such as Q-learning to On-policy methods such as Sarsa for the robustness reasons mentioned just above.

Thus, in the next section, we will focus on Model-Free methods based on a Off-Policy algorithm such as Q-learning.

2.2 Continuous Q-learning: a restricted state-of-the-art

In the previous section, we have examined RL methods in discrete state and action spaces. When states and actions become continuous, the standard adaptation of the RL methods consists in choosing a class of approximation functions to represent the continuous value function and in tuning their parameters so as to match the values known from single experiences. Tuning the parameters can be done using a Widrow-Hoff update rule on the gradient of the Q-function [1]:

$$\forall i \in [0, N], \Delta\omega_i = \alpha\delta_{QL}\frac{\partial Q(s_t, a_t)}{\partial \omega_i}$$

But Baird has shown that these methods, though they are fast, may not converge [18]. Instead, he shows the convergence of *residual gradient methods* based on the gradient of the TD error. We set $b = \arg\max a' Q(s_{t+1}, a')$ and we have:

$$\forall i \in [0, N], \Delta\omega_i = -\alpha\delta_{QL}[\frac{\partial \delta_{QL}}{\partial \omega_i}] = \alpha\delta_{QL}[\frac{\partial Q(s_t, a_t)}{\partial \omega_i} - \gamma\frac{\partial Q(s_{t+1}, b)}{\partial \omega_i}]$$

Once the tuning method is chosen, we can examine the classes of approximation functions that have been proposed to apply Off-policy RL algorithms to continuous state and action spaces. We will do so thanks to the following list of relevant properties inspired from [16], but adapted to our particular context.

Generalization ability Generalization comes from the ability of approximation functions to give a correct value in states that the agent never experienced before.

Model-independence To match our application requirement, the method must not assume the availability of the transition and reward functions in any way.

Fast retrieval of the best action (FRBA) Given that the state and action are continuous, finding the best action for a particular state given the different approximation function may be computationally expensive if the data structures are heavy. This may be incompatible with the real-time requirement of video games.

Continuity Since our application is navigation, we want our system to give very similar actions as output from very similar states.

[1] We note δ_{QL} the temporal difference error computed by Q-learning

Locality If the system learns something around a particular state, we do not want the modification to impair something learned elsewhere in the state space.

Readability The more readable the output of the learning process is, the easier it is to debug and reuse the knowledge expressed.

All systems studied in our state-of-the-art rely on an approximation of the Q-function, thus they all benefit from a generalization ability. Furthermore, we restrict the study to Model-independent systems.

CMAC Introduced by [19], the Cerebellar Model Articulation Controller (CMAC) discretizes a continuous state and action space into N overlapping partitions. Each element of each partition approximates the Q-function. The global value of a given state is the sum of the values given by all partitions. Being one of the earliest systems, CMAC is a reference in the domain, but most recent systems have a better performance and more interesting properties [3].

Learning Classifier Systems LCSs are rule-based systems combining RL algorithms with Genetic Algorithms. XCS, the most studied LCS so far, has recently been extended to deal with continuous states [20] and actions [21]. The main advantages of LCSs are their readability and the fact that learning is local. But they are very inefficient at retrieving the best action and the continuity property is not guaranteed since two different rules with similar conditions can trigger very different actions.

Discrete actions Neural Q-learning One way to deal with continuous states and actions consists in using a Neural Network (NN) with one input unit per perception and one output unit per action. The triggered action at each time step is the one whose corresponding output unit is the most activated. Some of these systems have been trained with Q-learning [22, 23]. The problem is that the more actions are needed, the bigger the network must be. Moreover, like most NN systems, this approach fails on the locality and readability properties, particularly in the case of multi-layer perceptrons. The case of RBF networks is particular and, in fact, much closer to our model presented in section 3, but we will not expand on that topic here.

Continuous actions Neural Q-learning Among NN approaches combining Q-learning with continuous states and actions, Dynamic Neural Fields [24] and Q-Kohon, a kind of Self Organizing Map, [25] do not match our needs because they are slow to reach a steady state, hence to provide the best action.

The *Wire Fitting Neural Network* technique [16] is of much more interest to us. It combines a single layer perceptron approximating the Q-function in any state with a Wire Fitting interpolator [26] dedicated to the fast retrieval of the best action. This system matches almost all our needs but fails on locality and readability, as all NN systems do.

Lazy Learning Lazy Learning methods are also called memory-based methods because they try to approximate the value function or the Q-function thanks to a rather simple storage of previous experiences of the agent, called "cases".

Previous cases below a given distance τ_s to the current state of the agent are "activated", thus used to interpolate the Q-function at that state, using a Locally Weighted Average (LWA) or a Locally Weighted Regression (LWR) [27].

A new case is stored each time there is no case to be activated below a distance Δ_s. Thus, Δ_s controls the density of cases (see figure 1).

There are two main instances of Lazy Learning methods. The instance-based learning algorithm HEDGER [4] works directly in a joint state and action space and stores simple (s_i, a_i, q_i) triples in each case.

The case-based learning algorithm from [3] is more complex. First, it is based on a Sarsa algorithm with eligibility traces (noted e_i hereafter) rather than on Q-learning. Second, the structure of each case C_i is noted $(s_i, Q_i, e_i, (a_j, q_{ij}, e_{ij})_{j=1...N})$, where the vector $(a_j)_{j=1...N}$ discretizes the action space. Third, instead of working in a joint $S \times A$ space, it first applies a distance function d^s between states and then a distance function d^a between action to determine the influence of each previous case in the interpolation. Fourth, the update rule combines an update of the average Q-value over all actions (with a coefficient ρ) and an update of the Q-values q_i of the individual actions a_i (with a coefficient (1-ρ)) as follows:

$$Q(s_t, a_t) = \sum_{C_i \in NN_t} \frac{K^s(d^s_{it})}{\sum_m K^s(d^s_{mt})} [\rho \sum_{a_j \in C_i} \frac{K^a(d^a_{jt})}{\sum_m K^a(d^a_{mt})} q_{ij} + (1-\rho) Q_i] \quad (5)$$

where K^s and K^a are kernel functions determining the influence of each previous experience depending on the distances, d^s_{it} is the distance between the states s_t and s_i and d^a_{jt} is the distance between actions a_j and a_t (see figure 1).

Both Lazy Learning approaches are local and continuous. But they do not provide a fast retrieval of the best action and their output is not easily readable.

2.3 Discussion

property	FRBA	Continuity	Locality	Readability
CMAC			X	
Learning Classifier Systems			X	X
Discrete actions Neural Q-learning	X			
Continuous actions Neural Q-learning	X	X		
Lazy learning		X	X	

Table 1. Synthesis of our state-of-the-art

From table 1, one can see that several choices are possible. But in [28], Baird claimed that the *Wire Fitting* techniques used by [16] could also be used in the

context of Lazy Learning methods, which makes this category more attractive than the others. In the next section, we present our adaptation of the case-based algorithm from [3] to our context.

3 Our model

Fig. 1. Illustration of the *case-based* approach in 2D. Points C1 to C6 are previous cases. Cases C1, C3 and C4 are activated in the computation of the value at the current state (black dot). The influence of each case in this computation is modulated by the kernel functions K^s and K^a shown on the left-hand side part of the figure. Updates are applied to all active cases. For more details, see [3].

Our model is similar to the one from [3], but differs on several respects:

- Rather than a Sarsa algorithm with eligibility traces, we use a residual gradient version of Q-learning;
- We set $\rho = 1$ in equation 5; as a result, the Q_i terms disappear of equation 5 ;
- The initialization of Q-values is not clearly specified in [3], thus we initialize the new case C_{new} as $(s_{new}, (a_j, q_{newj})_{j=1...N})$, where $s_{new} = s_t$ is the new state, and for each action a_j, we set $q_{newj} = R_{new} \frac{K^a(d^a_{jt})}{\sum_m K^a(d^a_{mt})}$
- Rather than a plain Euclidean distance, we use a weighted distance which takes into account the range of values of all dimensions in order to normalize their relative influence.

As a consequence of the first modifications, the structure of cases is much simplified, coming down to $(s_i, (a_j, q_{ij})_{j=1...N})$. In experiments not described here, we checked that all the simplifications above had no negative impact on the performance with respect to the system described by [3].

4 Experimental study

4.1 The *Mountain-Car* problem

The *Mountain-Car* problem is a classical benchmark for continuous state RL algorithms where a car must reach the top of a hill at the lowest possible speed. We follow the specification of [4]. The authors indicate that, with a fine-grained discretized tabular Q-learning, the average number of steps to the goal converges slowly to 56 steps from 2500 initial positions (see figure 2).

4.2 Experimental results and Discussion

Fig. 2. Comparison of the average number of steps necessary to reach the target position from 2500 initial positions, with tabular Q-learning (top), our *case-based* approach (bottom) and that of instance-based HEDGER (the latter reproduced from [4]). Results are obtained from a greedy policy. Between each performance measurement, a learning episode consists in initializing the car randomly and allowing a maximum duration of 200 time steps. Results are shown in steps rather than episodes here to give a more detailed view.

Figure 2 shows that, without any specific optimization, our approach converges much faster and towards a better performance than HEDGER.

In [4], the authors propose a series of optimizations that significantly improve the performance of HEDGER at such a point that their system finally performs better than ours. In particular, they use a form of prioritized sweeping which accelerates the learning process, but at the double cost of having to wait for the end of an episode and storing all data during an episode. In video games, an episode is virtually infinite, thus this method cannot be applied as such.

Thus our claim is that our approach is more suited to real-time and highly non-stationary environments such as video game than HEDGER, both because

our case-based approach implies much less memory requirements and because in such contexts it is more important to converge fast towards a reasonably efficient policy rather than converging more slowly to a policy that would have been better if the environment had not changed in between.

In order to further validate this claim, we check with additional experiments that our algorithm is able to deal efficiently with two simple forms of non-stationarities.

The first one is a sudden change in the dynamics of the environment, such that the value function must be completely learned again. In order to test this situation, we reverse the engine of the car each 5000 episodes, so that breaking becomes accelerating and *vice versa*. The second one is a slow drift of the dynamics of the environment. Here, the learner must responsively adapt its estimation of the value function while this function is changing. In order to test this situation, we let the amplitude of the acceleration of the car A_t evolve according to the equation $A_t = A_0(1 + 0,5sin(\frac{2\pi t}{T}))$ where A_0 is the standard amplitude and T corresponds to approximately 6000 learning episodes. Results are shown on figures 3(a) and 3(b).

(a) (b)

Fig. 3. (a) Performance when breaking and accelerating are reversed each 5000 episodes. (b) Performance when the acceleration amplitude is modified over time. Experimental conditions are the same as in figure 2. When the acceleration amplitude is the smallest, the car can hardly reach the top of the hill.

One can see that, in each context, the system is able to modify the policy quickly so as to adapt the behavior of the agent to the varying environment. Nevertheless, in the context of a radical and sudden change, it takes about 100 learning episodes before the policy gets efficient again. In the context of a slow drift, the performance gets poorer than in the stationary case when the acceleration is too weak to compensate for gravity, but the general loss of performance is small. Furthermore, we checked that we can reduce T to approximately 1000 learning episodes without further loss of performance. Thus the main concern given our target application can be expressed as two questions:

- are all the forms of non-stationarities that we will meet in video games approximated well enough with both forms studied here?
- will these non-stationarities be "nice" enough so that the loss of performance of the agent that they imply will stay acceptable in terms of observed behavior?

It is clear that some forms of non-stationarities due to the presence of other agents or to changes in the reward function have not been tested in this paper. But we hope that the ability to converge fast is a general answer to all these different questions, that would require different treatments if they were to be dealt with explicitly. This remains to be checked empirically.

5 Conclusion and Future work

In the case of a highly non-stationary and continuous environment such as a video game, rather than trying to deal explicitly with the non-stationarity with *ad hoc* methods, our approach in this paper has been to look for an algorithm that converges fast enough towards a correct policy so that the behavior can be adapted at any moment to the varying contexts.

Our experiments have highlighted the plausibility of this approach in the context of a well-known benchmark problem. The next thing to do now is to test our approach in the context of a video games to check if our basic assumptions are verified when confronted to the real difficulty.

In particular, we must check that, in the context of the *Mountain-Car* problem, there were only three discrete actions. We will have to check how our model behaves in the context of continuous actions implied by navigation problems. On a longer term, the result of this research will have to be integrated with the previous work of Robert [1] so as to combine the strengths of both contributions.

References

1. Robert, G.: MHiCS, une architecture de sélection de l'action Motivationnelle et Hiérarchique à Systèmes de Classeurs pour Personnages Non Joueurs adaptatifs. PhD thesis, Laboratoire d'Informatique de Paris 6 (2005)
2. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimal cost paths. IEEE transactions on SSC **4** (1968) 100–107
3. Santamaria, J.C., Sutton, R., Ram, A.: Experiments with reinforcement learning in problems with continuous state and action spaces. Adaptive Behavior **6** (1997) 163–218
4. Smart, W.D., Kaelbling, L.P.: Practical reinforcement learning in continuous spaces. In: 17th International Conference on Machine Learning. (2000) 903–910
5. Bertsekas, D.P.: Dynamic Programming and Optimal Control. Athena Scientific, Belmont, MA (1995)
6. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton, NJ (1957)
7. Bellman, R.E.: Adaptive Control Processes: A Guided Tour. Princeton University Press (1961)

8. Puterman, M.L., Shin, M.C.: Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. Management Science **24** (1978) 1127–1137
9. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
10. Watkins, C.J.C.H.: Learning with Delayed Rewards. PhD thesis, Psychology Department, University of Cambridge, England (1989)
11. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8** (1992) 279–292
12. Singh, S.P., Jaakkola, T., Littman, M.L., Szepesvari, C.: Convergence Results for Single-Step On-Policy Reinforcement Learning Algorithms. Machine Learning **38** (2000) 287–308
13. Sutton, R.S.: Learning to Predict by the Method of Temporal Differences. Machine Learning **3** (1988) 9–44
14. Jaakkola, T., Jordan, M.I., Singh, S.P.: On the Convergence of Stochastic Iterative Dynamic Programming Algorithms. Neural Computation **6** (1994) 1283–1288
15. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research **4** (1996) 237–285
16. Gaskett, C.: Q-Learning for Robot Control. PhD thesis, Australian National University (2002)
17. Sutton, R.S.: Dyna, an integrated architecture for learning, planning and reacting. SIGART Bulletin **2** (1991) 160–163
18. Baird, L.: Residual Algorithms: Reinforcement Learning with Function Approximation. In: Proceedings of the 12th International Conference on Machine Learning, San Francisco, CA, Morgan Kaufman Publishers (1995) 30–37
19. Albus, J.S.: A New Approach to Manipulator Control: the cerebellar model articulation controller (cmac). Journal of Dynamic Systems, Measurement and Control **97** (1975) 220–227
20. Wilson, S.W.: Get real! xcs with continuous-valued inputs. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Learning Classifier Systems. From Foundations to Applications. Springer, Berlin (2000) 209–219
21. Wilson, S.W.: Classifier systems for continuous payoff environments. In: GECCO. (2004) 824–835
22. Lin, L.J.: Self-Improving Reactive Agents based on Reinforcement Learning, Planning and Teaching. Machine Learning **8** (1992) 293–321
23. Werbos, P.J.: Approximate Dynamic Programming for Real-Time Control and Neural Modelling. In White, D., Solge, D., eds.: Handbook of Intelligent Control. Van Nostrand Reinhold, New York, NY (1992) 493–525
24. Gross, H.M., Stephan, V., Krabbes, M.: A Neural Field Approach to Topological Reinforcement Learning in Continuous Action Spaces. In: Proceedings of the IEEE World Congress on Computational Intelligence, IEEE Computer Society Press (1998) 1992–1997
25. Touzet, C.: Neural Reinforcement Learning for Behaviour Synthesis. Robotics and Autonomous Systems: Special Issue on Learning Robots: the New Wave **22** (1997) 251–281
26. Baird, L., Klopf, A.H.: Reinforcement Learning with High-Dimensional Continuous Actions. Technical Report WL-TR-93-1147, Wright-Patterson Air Force Base Ohio (1993)
27. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally Weighted Learning. Artificial Intelligence Review **11** (1996) 11–73
28. Baird, L., Moore, A.W.: Gradient Descent for General Reinforcement Learning. In: Advances in Neural Information Processing Systems 11, MIT Press (1999) 968–974