Continuous-State Reinforcement Learning with Fuzzy Approximation

Lucian Buşoniu[1], Damien Ernst[2], Bart De Schutter[1], and Robert Babuška[1]

[1] Delft University of Technology, The Netherlands
[2] Supélec, Rennes, France
i.l.busoniu@tudelft.nl, damien.ernst@supelec.fr, b@deschutter.info, r.babuska@tudelft.nl

Abstract. Reinforcement learning (RL) is a widely used learning paradigm for adaptive agents. There exist several convergent and consistent RL algorithms which have been intensively studied. In their original form, these algorithms require that the environment states and agent actions take values in a relatively small discrete set. Fuzzy representations for approximate, model-free RL have been proposed in the literature for the more difficult case where the state-action space is continuous. In this work, we propose a fuzzy approximation architecture similar to those previously used for Q-learning, but we combine it with the model-based Q-value iteration algorithm. We prove that the resulting algorithm converges. We also give a modified, asynchronous variant of the algorithm that converges at least as fast as the original version. An illustrative simulation example is provided.

1 Introduction

Learning agents can tackle problems that are difficult to solve with pre-programmed solutions. Reinforcement learning (RL) is a popular learning paradigm for adaptive agents, thanks to its mild assumptions on the environment (which can be a nonlinear, stochastic process), and thanks to its ability to work without an explicit model of the environment [1,2]. At each time step, an RL agent perceives the complete state of the environment and takes an action. This action causes the environment to move into a new state. The agent then receives a scalar reward signal indicating the quality of this transition. The agent's objective is to maximize the cumulative reward over the course of interaction. There exist several convergent and consistent RL algorithms which have been intensively studied [1,2,3]. Unfortunately, these algorithms apply in general only to problems having discrete and not too large state-action spaces since, among others, they require to store estimates of cumulative rewards for every state or state-action pair. For problems with discrete but large state-action spaces, or continuous state-action spaces, approximate algorithms have to be used.

In this paper, we analyze the convergence of some model-based reinforcement learning algorithms exploiting a fuzzy approximation architecture. Our algorithms deal with problems for which the complexity comes from the state space, but not the action space, i.e., where the state space contains an infinite or extremely large number of elements and the action space is discrete and moderate in size. Most of our results also hold in the case where the action space is large or continuous, but in that case require a discretization procedure that selects a moderate number of representative actions. A significant number of related fuzzy approximators have been proposed, e.g., for Q-learning [4, 5, 6] or actor-critic algorithms [6, 7, 8, 9, 10]. However, most of these approaches are heuristic in nature, and their theoretical properties have not been investigated. Notable exceptions are the actor-critic algorithms in [8, 9].

On the other hand, a rich body of literature concerns the theoretical analysis of approximate RL algorithms, both in a model-based setting [11, 12, 13, 14] and when an *a priori* model is not available [15, 16, 17, 18, 19].[1] While convergence is not ensured for an arbitrary approximator (see [11, 14] for examples of divergence), there exist approximation schemes that do provide convergence guarantees. These mainly belong to the family of linear basis functions, and are encountered under several other names: kernel functions [15, 16], averagers [13], interpolative representations [17]. Some authors also investigate approximators that alter their structure during learning in order to better represent the solution [16, 20, 21]. While some of these algorithms exhibit impressing learning capabilities, they may face convergence problems [16].

Here, we consider an approximator that represents the Q-function using a fuzzy partition of the state space. While similar representations have previously been used in fuzzy Q-learning, in this paper the fuzzy approximator is combined with the model-based Q-value iteration algorithm. The resulting algorithm is shown to converge. Afterwards, we propose a variant of this algorithm, which we name asynchronous fuzzy Q-iteration, and which we show converges at least as fast as the original version. Asynchronous Q-iteration has been widely used in exact RL, but its approximate counterpart has not been studied before.

The remainder of this paper is structured as follows. Section 2 describes briefly the RL problem and reviews some relevant results from the dynamic programming theory. Section 3 introduces the approximate Q-iteration algorithm, which is an extension of the classical Q-iteration algorithm to cases where function approximators are used. Section 4 presents the proposed fuzzy approximator. The properties of synchronous and asynchronous approximate Q-iteration using this approximator are analyzed in Section 5. Section 6 applies the algorithms introduced to a nonlinear control problem with four continuous state variables, and compares the performance of the algorithms with that of Q-iteration with radial basis function approximation. Section 7 describes possible extensions of the algorithm for stochastic tasks and online learning. Section 8 outlines ideas for future work and concludes the paper.

[1] Some authors use 'model-based RL' when referring to algorithms that build a model of the environment from interaction. We use the term 'model-learning' for such techniques, and reserve the name 'model-based' for algorithms that rely on an *a priori* model of the environment.

2 Reinforcement Learning

In this section, the RL task is briefly introduced and its optimal solution is characterized. The presentation is based on [1, 2].

Consider a deterministic *Markov decision process (MDP)* with the state space X, the action space U, the transition function $f : X \times U \to X$, and the reward function $\rho : X \times U \to \mathbb{R}$.[2] As a result of the agent's action u_k in state x_k at the discrete time step k, the state changes to $x_{k+1} = f(x_k, u_k)$. At the same time, the agent receives the scalar reward signal $r_{k+1} = \rho(x_k, u_k)$, which evaluates the immediate effect of action u_k, but says nothing about its long-term effects.[3]

The agent chooses actions according to its policy $h : X \to U$, using $u_k = h(x_k)$. The goal of the agent is to learn a policy that maximizes, starting from the current moment in time ($k = 0$) and from any state x_0, the discounted return:

$$R = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k) \tag{1}$$

where $\gamma \in [0, 1)$ and $x_{k+1} = f(x_k, u_k)$ for $k \geq 0$. The discounted return compactly represents the reward accumulated by the agent in the long run. The learning task is therefore to maximize the long-term performance, while only receiving feedback about the immediate, one-step performance. This can be achieved by computing the optimal action-value function.

An action-value function (Q-function), $Q^h : X \times U \to \mathbb{R}$, gives the return of each state-action pair under a given policy h:

$$Q^h(x, u) = \rho(x, u) + \sum_{k=1}^{\infty} \gamma^k \rho(x_k, h(x_k)) \tag{2}$$

where $x_1 = f(x, u)$ and $x_{k+1} = f(x_k, h(x_k))$ for $k \geq 1$. The optimal action-value function is defined as $Q^*(x, u) = \max_h Q^h(x, u)$. Any policy that picks for every state the action with the highest optimal Q-value:

$$h^*(x) = \arg\max_u Q^*(x, u) \tag{3}$$

is optimal, i.e., it maximizes the return (1).

A central result in RL, upon which many algorithms rely, is the *Bellman optimality equation*:

$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u' \in U} Q^*(f(x, u), u') \quad \forall x, u \tag{4}$$

[2] Throughout the paper, the standard control-theoretic notation is used: x for state, X for state space, u for control action, U for action space, f for environment dynamics. We denote reward functions by ρ, to distinguish them from the instantaneous rewards r and the return R. We denote policies by h.

[3] A stochastic formulation is possible. In that case, expected returns under the probabilistic transitions must be considered.

This equation can be solved using the Q-value iteration algorithm. Let the set of all Q-functions be denoted by $\mathcal{Q}$. Define the Q-iteration mapping $T : \mathcal{Q} \to \mathcal{Q}$, which computes the right-hand side of the Bellman equation for any Q-function:

$$[T(Q)](x,u) = \rho(x,u) + \gamma \max_{u' \in U} Q(f(x,u), u') \tag{5}$$

Using this notation, the Bellman equation (4) states that Q^* is a fixed point of T, i.e., $Q^* = T(Q^*)$. The following result is also well-known.

Theorem 1. *T is a contraction with factor $\gamma < 1$ in the infinity norm, i.e., for any pair of functions Q and Q', it is true that $\|T(Q) - T(Q')\|_\infty \leq \gamma \|Q - Q'\|_\infty$.*

The Q-value iteration (Q-iteration, for short) algorithm starts from an arbitrary Q-function Q_0 and in each iteration κ updates the Q-function using the formula $Q_{\kappa+1} = T(Q_\kappa)$. From Theorem 1, it follows that T has a unique fixed point, and since from (4) this point is Q^*, the iterative scheme converges to Q^* as $\kappa \to \infty$.

Q-iteration uses an *a priori* model of the task, in the form of the transition and reward functions f, ρ. There also exist algorithms that do not require an *a priori* model. Model-free algorithms like Q-learning work without an explicit model, by learning directly the optimal Q-function from real or simulated experience in the environment. Model-learning algorithms like Dyna estimate a model from experience and use it to derive Q^* [1].

3 Q-iteration with Function Approximation

In general, the implementation of Q-iteration (5) requires that Q-values are stored and updated explicitly for each state-action pair. If some of the state or action variables are continuous, the number of state-action pairs is infinite, and an exact implementation is impossible. Instead, approximate solutions must be used. Even if the number of state-action pairs is finite but very large, exact Q-iteration might be impractical, and it is useful to approximate the Q-function.

The following mappings are defined in order to formalize approximate Q-iteration (the notation follows [17]).

1. The *Q-iteration* mapping T, defined by equation (5).
2. The *approximation* mapping $F : \mathbb{R}^n \to \mathcal{Q}$, which for a given value of the parameter vector $\theta \in \mathbb{R}^n$ produces an approximate Q-function $\widehat{Q} = F(\theta)$. In other words, the parameter vector θ is a finite representation of $\widehat{Q}$.
3. The *projection* mapping $P : \mathcal{Q} \to \mathbb{R}^n$, which given a target Q-function Q computes the parameter vector θ such that $F(\theta)$ is as close as possible to Q (e.g., in a least-squares sense).

The notation $[F(\theta)](x,u)$ refers to the value of the Q-function $F(\theta)$ for the state-action pair (x,u). The notation $[P(Q)]_l$ refers to the l-th component in the parameter vector $P(Q)$.

Approximate Q-iteration starts with an arbitrary (e.g., identically 0) parameter vector θ_0 and at each iteration κ updates it using the composition of the mappings P, T, and F:

$$\theta_{\kappa+1} = PTF(\theta_\kappa) \tag{6}$$

Unfortunately, the approximate Q-iteration is not guaranteed to converge for an arbitrary approximator. Counter-examples can be found for the related value-iteration algorithm (e.g., [11]), but they apply directly to the Q-iteration algorithm, as well. One particular case in which approximate Q-iteration converges is when the composite mapping PTF is a contraction [11,13]. This property will be used below to show that fuzzy Q-iteration converges.

4 Fuzzy Q-iteration

In this section, we propose a fuzzy approximation architecture similar to those previously used in combination with Q-learning [4,6], and apply it to the model-based Q-iteration algorithm. The theoretical properties of the resulting fuzzy Q-iteration algorithm are investigated in Section 5.

In the sequel, it is assumed that the action space is discrete. We denote it by $U_0 = \{u_j | j = 1, \dots, M\}$. For instance, this discrete set can be obtained from the discretization of an originally continuous action space. The state space can be either continuous or discrete. In the latter case, fuzzy approximation is useful when the number of discrete states is large.

The proposed approximation scheme relies on a fuzzy partition of the state space into N sets $\mathcal{X}_i$, each described by a membership function $\mu_i : X \to [0, 1]$. A state x belongs to each set i with a degree of membership $\mu_i(x)$. In the sequel the following assumptions are made:

1. The fuzzy partition has been normalized, i.e., $\sum_{i=1}^{N} \mu_i(x) = 1$, $\forall x \in X$.
2. All the fuzzy sets in the partition are normal and have singleton cores, i.e., for every i there exists a unique x_i for which $\mu_i(x_i) = 1$ (consequently, $\mu_{\underline{i}}(x_i) = 0$ for all $\underline{i} \neq i$ by Assumption 1). The state x_i is called the core (center value) of the set $\mathcal{X}_i$. This second assumption is required here for brevity in the description and analysis of the algorithms; it can be relaxed using results of [11].

For two examples of fuzzy partitions that satisfy the above conditions, see Figure 1, from Section 6.

The fuzzy approximator stores an $N \times M$ matrix of parameters, with one component $\theta_{i,j}$ corresponding to each core-action pair (x_i, u_j).[4] The approximator takes as input a state-action pair (x, u_j) and outputs the Q-value:

$$\widehat{Q}(x, u_j) = [F(\theta)](x, u_j) = \sum_{i=1}^{N} \mu_i(x)\theta_{i,j} \tag{7}$$

[4] The matrix arrangement is adopted for convenience of notation only. For the theoretical study of the algorithms, the collection of parameters is still regarded as a vector, leading e.g., to $\|\theta\|_\infty = \max_{i,j} |\theta_{i,j}|$.

Algorithm 1. Synchronous fuzzy Q-iteration

```
1: $\theta_0 \leftarrow 0$; $\kappa \leftarrow 0$
2: repeat
3:     for $i = 1, \ldots, N, j = 1, \ldots, M$ do
4:         $\theta_{\kappa+1,i,j} \leftarrow \rho(x_i, u_j) + \gamma \max_{\underline{j}} \sum_{\underline{i}=1}^{N} \mu_{\underline{i}}(f(x_i, u_j))\theta_{\kappa,\underline{i},\underline{j}}$
5:     end for
6:     $\kappa \leftarrow \kappa + 1$
7: until $\|\theta_\kappa - \theta_{\kappa-1}\|_\infty \leq \delta$
```

This is a linear basis-functions form, with the basis functions only depending on the state. The approximator (7) can be regarded as M distinct approximators, one for each of the M discrete actions.

The projection mapping infers from a Q-function the values of the approximator parameters according to the relation:

$$\theta_{i,j} = [P(Q)]_{i,j} = Q(x_i, u_j) \tag{8}$$

Note this is the solution θ to the problem:

$$\sum_{i=1,\ldots,N, j=1,\ldots,M} |[F(\theta)](x_i, u_j) - Q(x_i, u_j)|^2 = 0$$

The approximator (7), (8) shares some strong similarities with several classes of approximators that have already been used in RL: interpolative representations [11], averagers [13], and representative-state techniques [19].

The Q-iteration algorithm using the approximation mapping (7) and projection mapping (8) can be written as Algorithm 1. To establish the equivalence between Algorithm 1 and the approximate Q-iteration in the form (6), observe that the right-hand side in line 4 of Algorithm 1 corresponds to $[T(\widehat{Q}_\kappa)](x_i, u_j)$, where $\widehat{Q}_\kappa = F(\theta_\kappa)$. Hence, line 4 can be written $\theta_{\kappa+1,i,j} \leftarrow [PTF(\theta_\kappa)]_{i,j}$ and the entire **for** loop described by lines 3–5 is equivalent to (6).

Algorithm 2 is a different version of fuzzy Q-iteration, that makes more efficient use of the updates by using the latest updated values of the parameters θ in each step of the computation. Since the parameters are updated in an asynchronous fashion, this version is called *asynchronous Q-iteration* (in Algorithm 2 parameters are updated in sequence, but they can actually be updated in any order and our results still hold). Although the exact version of asynchronous Q-iteration is widely used [1, 2], the asynchronous variant has received little attention in the context of approximate RL. To differentiate between the two versions, Algorithm 1 is hereafter called *synchronous fuzzy Q-iteration*.

5 Convergence of Fuzzy Q-iteration

In this section, the convergence of synchronous and asynchronous fuzzy Q-iteration is established, i.e., it is shown that there exists a parameter vector θ^* such that for both algorithms, $\theta_\kappa \to \theta^*$ as $\kappa \to \infty$. In addition, asynchronous

Algorithm 2. Asynchronous fuzzy Q-iteration

```
1: $\theta_0 \leftarrow 0$; $\kappa \leftarrow 0$
2: repeat
3:     $\theta \leftarrow \theta_\kappa$
4:     for $i = 1, \dots, N, j = 1, \dots, M$ do
5:         $\theta_{i,j} \leftarrow \rho(x_i, u_j) + \gamma \max_{\underline{j}} \sum_{\underline{i}=1}^{N} \mu_{\underline{i}}(f(x_i, u_j))\theta_{\underline{i},\underline{j}}$
6:     end for
7:     $\theta_{\kappa+1} \leftarrow \theta$
8:     $\kappa \leftarrow \kappa + 1$
9: until $\|\theta_\kappa - \theta_{\kappa-1}\|_\infty \le \delta$
```

fuzzy Q-iteration is shown to converge at least as fast as the synchronous version. The distance between $F(\theta^*)$ and the true optimum Q^*, as well as the suboptimality of the greedy policy in $F(\theta^*)$, are also shown to be bounded [11,13]. The consistency of fuzzy Q-iteration, i.e., the convergence to the optimal Q-function Q^* as the maximum distance between the cores of adjacent fuzzy sets goes to 0, is not studied here, and is a topic for future research.

Proposition 1. *Synchronous fuzzy Q-iteration (Algorithm 1) converges.*

Proof. The proof follows from the proof of convergence of (synchronous) value iteration with averagers [13], or with interpolative representations [11]. This is because fuzzy approximation is an averager by the definition in [13], and an interpolative representation by the definition in [11]. The main idea of the proof is that PTF is a contraction with factor $\gamma < 1$, i.e., $\|PTF(\theta) - PTF(\theta')\|_\infty \le \gamma \|\theta - \theta'\|_\infty$, for any θ, θ'. This is true thanks to the non-expansive nature of P and F, and because T is a contraction. □

It is shown next that asynchronous fuzzy Q-iteration (Algorithm 2) converges. The convergence proof is similar to that for *exact* asynchronous value iteration [2].

Proposition 2. *Asynchronous fuzzy Q-iteration (Algorithm 2) converges.*

Proof. Denote $n = N \cdot M$, and rearrange the matrix θ into a vector in $\mathbb{R}^n$, placing first the elements of the first row, then the second etc. The element at row i and column j of the matrix is now the l-th element of the vector, with $l = (i-1) \cdot M + j$.

Define for all $l = 0, \dots, n$ recursively the mappings $S_l : \mathbb{R}^n \to \mathbb{R}^n$ as:

$$S_0(\theta) = \theta, \quad [S_l(\theta)]_{\underline{l}} = \begin{cases} [PTF(S_{l-1}(\theta))]_{\underline{l}} & \text{if } \underline{l} = l \\ [S_{l-1}(\theta)]_{\underline{l}} & \text{if } \underline{l} \in \{1, \dots, n\} \setminus l \end{cases}$$

In words, S_l corresponds to updating the first l parameters using approximate asynchronous Q-iteration, and S_n is a complete iteration of the approximate asynchronous algorithm. Now we show that S_n is a contraction, i.e.,

$\|S_n(\theta) - S_n(\theta')\|_\infty \leq \gamma \|\theta - \theta'\|_\infty$, for any θ, θ'. This can be done element-by-element. By the definition of S_l, the first element is only updated by S_1:

$$\begin{aligned} |[S_n(\theta)]_1 - [S_n(\theta')]_1| &= |[S_1(\theta)]_1 - [S_1(\theta')]_1| \\ &= |[PTF(\theta)]_1 - [PTF(\theta')]_1| \\ &\leq \gamma \|\theta - \theta'\|_\infty \end{aligned}$$

The last step follows from the contraction mapping property of PTF.

Similarly, the second element is only updated by S_2:

$$\begin{aligned} |[S_n(\theta)]_2 - [S_n(\theta')]_2| &= |[S_2(\theta)]_2 - [S_2(\theta')]_2| \\ &= |[PTF(S_1(\theta))]_2 - [PTF(S_1(\theta'))]_2| \\ &\leq \gamma \|S_1(\theta) - S_1(\theta')\|_\infty \\ &= \gamma \max\{|[PTF(\theta)]_1 - [PTF(\theta')]_1|, \\ &\qquad |\theta_2 - \theta'_2|, \ldots, |\theta_n - \theta'_n|\} \\ &\leq \gamma \|\theta - \theta'\|_\infty \end{aligned}$$

where $\|S_1(\theta) - S_1(\theta')\|_\infty$ is expressed by direct maximization over its elements, and the contraction mapping property of PTF is used twice. Continuing in this fashion, we obtain $|[S_n(\theta)]_l - [S_n(\theta')]_l| \leq \gamma \|\theta - \theta'\|_\infty$ for all l, and thus S_n is a contraction. Therefore, asynchronous fuzzy Q-iteration converges. □

This proof is actually more general, showing that approximate asynchronous Q-iteration converges for any approximation mapping F and projection mapping P for which PTF is a contraction. It can also be easily shown that synchronous and fuzzy Q-iteration converge to the same parameter vector; indeed, the repeated application of any contraction mapping will converge to its unique fixed point regardless of whether it is applied in a synchronous or asynchronous (element-by-element) fashion.

We now show that asynchronous fuzzy Q-iteration converges at least as fast as the synchronous version. For that, we first need the following monotonicity lemma. In this lemma, as well as in the sequel, vector and function inequalities are understood to be satisfied element-wise.

Lemma 1 (Monotonicity of PTF). *If $\theta \leq \theta'$, then $PTF(\theta) \leq PTF(\theta')$.*

Proof. It will be shown in turn that F, T, and P are monotonous. To show that F is monotonous we show that, given $\theta \leq \theta'$, it follows that for all x, u_j:

$$[F(\theta)](x, u_j) \leq [F(\theta')](x, u_j) \quad \Leftrightarrow \quad \sum_{i=1}^{N} \mu_i(x)\theta_{i,j} \leq \sum_{i=1}^{N} \mu_i(x)\theta'_{i,j}$$

The last inequality is true by the assumption $\theta \leq \theta'$.

To show that T is monotonous we show that, given $Q \leq Q'$, it follows that:

$$\begin{aligned} &[T(Q)](x, u) \leq [T(Q')](x, u) \\ \Leftrightarrow\ &\rho(x, u) + \gamma \max_{u' \in U} Q(f(x, u), u') \leq \rho(x, u) + \gamma \max_{u' \in U} Q'(f(x, u), u') \\ \Leftrightarrow\ &\max_{u' \in U} Q(f(x, u), u') \leq \max_{u' \in U} Q'(f(x, u), u') \end{aligned}$$

The last inequality is true because $Q(f(x,u),u') \leq Q'(f(x,u),u')$ for all u', which follows from the assumption $Q \leq Q'$.

To show that P is monotonous we show that, given $Q \leq Q'$, it follows that for all i,j:

$$[P(Q)]_{i,j} \leq [P(Q')]_{i,j} \quad \Leftrightarrow \quad Q(x_i,u_j) \leq Q'(x_i,u_j)$$

The last inequality is true by assumption. Therefore, the composite mapping PTF is monotonous. □

Proposition 3. *If a parameter vector θ satisfies $\theta \leq PTF(\theta) \leq \theta^*$, then:*

$$(PTF)^k(\theta) \leq S^k(\theta) \leq \theta^* \quad \forall k \geq 1$$

Proof. This follows from the monotonicity of PTF, and can be shown element-wise, in a similar fashion to the proof of Proposition 2. Note that this result is an extension of Bertsekas' result on exact value iteration [2]. □

In words, Proposition 3 states that k iterations of asynchronous fuzzy Q-iteration move the parameter vector at least as close to the convergence point as k iterations of the synchronous algorithm.

The following bounds on the sub-optimality of the convergence point, and of the policy corresponding to this point, follow from [11, 13].

Proposition 4. *If the action space of the original problem is discrete and all the discrete actions are used for fuzzy Q-iteration, then the convergence point θ^* of asynchronous and synchronous fuzzy Q-iteration satisfies:*

$$\|Q^* - F(\theta^*)\|_\infty \leq \frac{2\varepsilon}{1-\gamma} \tag{9}$$

$$\|Q^* - Q^{\widehat{h}^*}\|_\infty \leq \frac{4\gamma\varepsilon}{(1-\gamma)^2} \tag{10}$$

where $\varepsilon = \min_{\underline{Q}} \left\|Q^* - \underline{Q}\right\|_\infty$ is the minimum distance between Q^* and any fixed point $\underline{Q}$ of the composite mapping FP, and $Q^{\widehat{h}^*}$ is the Q-function of the approximately optimal policy $\widehat{h}^*(x) = \arg\max_u [F(\theta^*)](x,u)$.

For example, any Q-function that satisfies $Q(x,u_j) = \sum_{i=1}^N \mu_i(x) Q(x_i,u_j)$ for all x,j is a fixed point of FP. In particular, if the optimal Q-function has this form, i.e., is exactly representable by the chosen fuzzy approximator, the algorithm will converge to it, and the corresponding policy will be optimal (since in this case $\varepsilon = 0$).

In this section, we have established fuzzy Q-value iteration as a theoretically sound technique to perform approximate RL in continuous-variable tasks. In addition to the convergence of both synchronous and asynchronous fuzzy Q-iteration, it was shown that the asynchronous version converges at least as fast as the synchronous one, and therefore might be more desirable in practice. When the action space is discrete, the approximation error of the resulting Q-function is bounded (9) and the sub-optimality of the resulting policy is also bounded (10) (the latter may be more relevant in practice). These bounds provide confidence in the results of fuzzy Q-iteration.

6 Example: 2-D Navigation

In this section, fuzzy Q-iteration is applied to a two-dimensional (2-D) simulated navigation problem with continuous state and action variables. A point-mass with a unit mass value (1kg) has to be steered on a rectangular surface such that it gets close to the origin in minimum time, and stays there. The state x contains the 2-D coordinates of the point mass, c_x and c_y, and its 2-D velocity: $x = [c_x, c_y, \dot{c}_x, \dot{c}_y]^{\mathrm{T}}$. The motion of the point-mass is affected by friction, which can vary with the position, making the dynamics non-linear. Formally, the continuous-time dynamics of this system are:

$$\begin{bmatrix} \ddot{c}_x \\ \ddot{c}_y \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} - b(c_x, c_y) \begin{bmatrix} \dot{c}_x \\ \dot{c}_y \end{bmatrix} \tag{11}$$

where the control input $u = [u_x, u_y]^{\mathrm{T}}$ is a 2-D force (acceleration), and the scalar function $b(c_x, c_y)$ is the position-dependent damping coefficient (friction). All the state and action variables are bounded. The bounds are listed in Table 1, along with the meaning and units of all the variables.

Table 1. Variables for the navigation problem

Symbol	Parameter	Domain; Unit
c_x, c_y	horizontal, vertical coordinate	$[-5, 5]$ m
$\dot{c}_x, \dot{c}_y$	horizontal, vertical velocity	$[-2, 2]$ m/s
u_x, u_y	horizontal, vertical control force	$[-1, 1]$ N
b	damping coefficient	$\mathbb{R}^+$ kg/s

To obtain the transition function f for RL, time is discretized with a step of $T_{\mathrm{s}} = 0.2$ s, and the dynamics (11) are numerically integrated between the sampling instants.[5] The goal of reaching the origin in minimum time is expressed by the following reward function:

$$\rho(x, u) = \begin{cases} 10 & \text{if } \|x\|_\infty \leq 0.2 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

This means that the coordinates and the velocities along each axis have to be smaller than 0.2 in magnitude for the agent to get a non-zero reward.

The control force is discretized into 9 discrete values: $U_0 = \{-1, 0, 1\} \times \{-1, 0, 1\}$. These correspond to full acceleration into the 4 cardinal directions, diagonally, and no acceleration at all. Each of the individual velocity domains is partitioned into a triangular fuzzy partition with three fuzzy sets centered at $\{-2, 0, 2\}$, as in Figure 1, left. Since the triangular partition satisfies Assumptions 1, 2, the set of cores completely determines the shape of the membership functions. Triangular partitions are also used for the position coordinates. Different partitions of the position variables are used for each of the two damping landscapes considered in the sequel.

[5] The numerical integration algorithm is the Dormand-Prince variant of Runge-Kutta, as implemented in the MATLAB 7.2 function `ode45`.

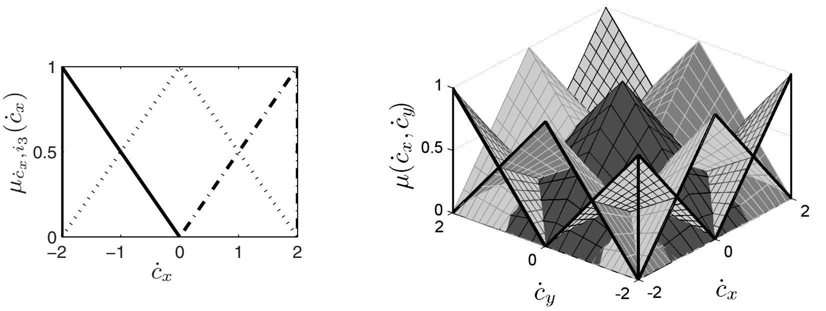

Fig. 1. *Left:* triangular fuzzy partition for $\dot{c}_x \in [-2,2]$. Each membership function is plotted in a different line style. The partition for $\dot{c}_y$ is identical. *Right:* composition of the fuzzy partitions for $\dot{c}_x, \dot{c}_y$, yielding the two-dimensional fuzzy partition for $[\dot{c}_x, \dot{c}_y]^{\mathrm{T}}$. Each membership surface is plotted in a different style. The original single-dimensional fuzzy partitions are highlighted in full black lines.

The fuzzy partition of the state space $X = [-5,5]^2 \times [-2,2]^2$ is then defined as follows. One fuzzy set is computed for each combination (i_1, i_2, i_3, i_4) of individual sets for the four state components: μ_{c_x,i_1}; μ_{c_y,i_2}; $\mu_{\dot{c}_x,i_3}$; and $\mu_{\dot{c}_y,i_4}$. The membership function of each composite set is defined as the product of the individual membership functions, applied to their individual variables:

$$\mu(x) = \mu_{c_x,i_1}(c_x) \cdot \mu_{c_y,i_2}(c_y) \cdot \mu_{\dot{c}_x,i_3}(\dot{c}_x) \cdot \mu_{\dot{c}_y,i_4}(\dot{c}_y) \tag{13}$$

where $x = [c_x, c_y, \dot{c}_x, \dot{c}_y]^{\mathrm{T}}$. It is easy to verify that the fuzzy partition computed in this way still satisfies Assumptions 1, 2. This way of building the state space partition can be thought of as a conjunction of one-dimensional concepts corresponding to the fuzzy partitions of the individual state variables. An example of such a composition for the two velocity variables is given in Figure 1, right.

6.1 Uniform Damping Landscape

In a first, simple scenario, the damping was kept constant: $b(c_x, c_y) = b_0 = 0.5\,\mathrm{kg/s}$. Identical triangular fuzzy partitions were defined for c_x and c_y, with the cores in $\{-5, -2, -0.3, -0.1, 0, 0.1, 0.3, 2, 5\}$. Asynchronous and synchronous fuzzy Q-iteration were run with the discount factor $\gamma = 0.98$ and the threshold $\delta = 0.01$ (see Algorithms 1 and 2). The parameters γ and δ are set somewhat arbitrarily, but their variation around the given values does not significantly affect the computed policy. The asynchronous algorithm converged in 339 iterations; the synchronous one in 343. Therefore, in this particular problem, the speed of convergence for the asynchronous algorithm is close to the speed for the synchronous one (i.e., the worst-case bound). The policies computed by the two algorithms are similar.

A continuous policy was obtained by interpolating between the best local actions, using the membership degrees as weights: $\widehat{h}^*(x) = \sum_{i=1}^{N} \mu_i(x) u_{j_i^*}$, where j_i^* is the index of the best local action for the core state x_i, $j_i^* = \arg\max_j [F(\theta^*)](x_i, u_j) = \arg\max_j \theta^*_{i,j}$.

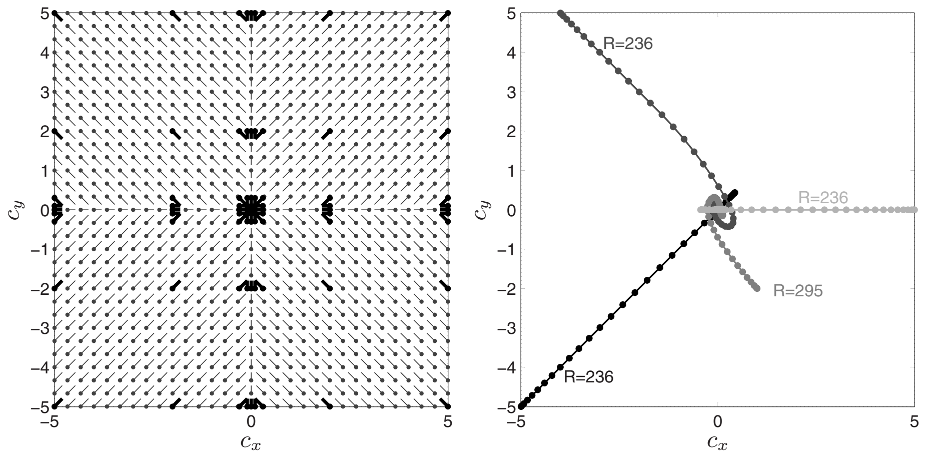

Fig. 2. *Left:* The policy for constant damping, when $\dot{c}_x = \dot{c}_y = 0$. The direction and magnitude of the control force in a grid of sample points (marked by dots) is indicated by lines. Thick, black lines indicate exact policy values in the cores of the fuzzy sets (marked by thick, black dots). *Right:* A set of representative trajectories, each given in a different shade of gray. The initial velocity is always zero. The position of the point-mass at each sample is indicated by dots. The closer the dots, the smaller the velocity is. The accumulated discounted return is displayed alongside each trajectory, rounded off toward zero.

Figure 2 presents a slice through the computed policy for zero velocity, $\widehat{h}^*(c_x, c_y, 0, 0)$, and plots some representative trajectories. The zero-velocity slice is clearly different from the optimal continuous-action policy, which would steer the agent directly towards the goal zone from any position. Also, since the actions are originally continuous, the bound (10) does not apply. Nevertheless, the zero-velocity slice presented in the figure is close to the best that can be achieved with the given action quantization.

6.2 Varying Damping Landscape

In the second scenario, to increase the difficulty of the problem, the damping (friction with the surface) varies as an affine sum of L Gaussian functions:

$$b(c_x, c_y) = b_0 + \sum_{i=1}^{L} b_i \exp\left[-\frac{(c_x - g_{x,i})^2}{\sigma_{x,i}^2} - \frac{(c_y - g_{y,i})^2}{\sigma_{y,i}^2}\right] \tag{14}$$

The chosen values were: $b_0 = 0.5$, $L = 2$, $b_1 = b_2 = 8$, $g_{x,1} = 0$, $g_{y,1} = -2.3$, $\sigma_{x,1} = 2.5$, $\sigma_{x,2} = 1.5$, and for the second Gaussian function: $g_{x,2} = 4.7$, $g_{y,2} = 1$, $\sigma_{x,2} = 1.5$, $\sigma_{y,2} = 2$. So, the damping is largest (around 8.5 kg/s) at positions $(0, -2.3)$ and $(4.7, 1)$. The damping variation can be observed in Figure 3, where the surface is colored darker when the damping is larger.

The fuzzy set cores for the position partition are marked by thick black dots in Figure 3, left. They were chosen based on prior knowledge about the

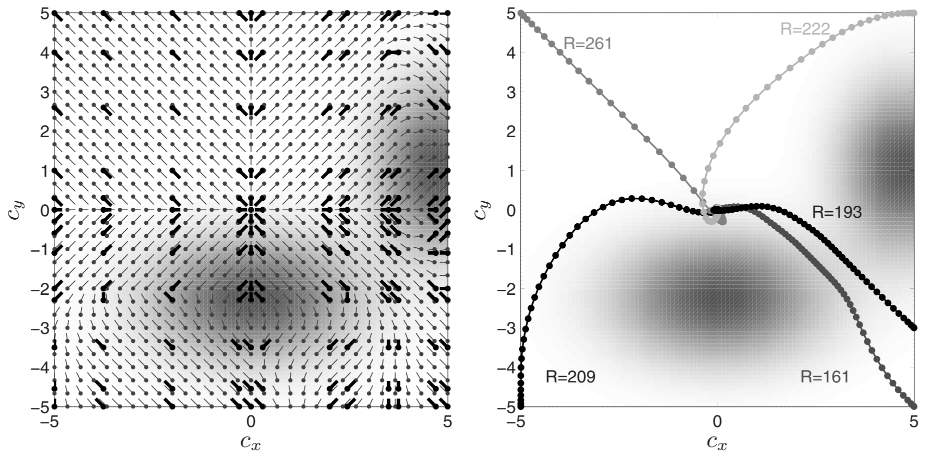

Fig. 3. *Left:* The policy for varying damping (14), when $\dot{c}_x = \dot{c}_y = 0$. Darker areas indicate larger damping. The direction and magnitude of the control force in a grid of sample points is indicated. The thick, black lines indicate exact policy values in the cores of the fuzzy partition. *Right:* A set of representative controlled trajectories with the associated returns.

position of the high-friction areas. The cores include representative points around these areas, and some points near the goal. Asynchronous and synchronous fuzzy Q-iteration were run with the same settings as before, and converged in the same number of iterations. Figure 2 presents a slice through the resulting policy for zero velocity, $\widehat{h}^*(c_x, c_y, 0, 0)$, together with several sample trajectories. It can be clearly seen how the policy steers around the high-friction areas, and how the interpolation helps in providing meaningful commands between the fuzzy cores.

6.3 Comparison with RBF Q-iteration

There exist other approximators than fuzzy partitions that could be combined with Q-iteration to yield convergent algorithms. These approximators are usually restricted classes of linear basis functions, satisfying conditions related to (but different from) Assumption 2 of Section 4. As space limitations do not allow for an extensive comparison of fuzzy Q-iteration with such algorithms, this section compares it with one of them, namely Q-iteration with normalized radial basis function (RBF) approximation.

Define a set of N normalized RBFs $\varphi_i : X \to \mathbb{R}$, $i = 1, \ldots, N$, as follows:

$$\varphi_i(x) = \frac{\underline{\varphi}_i(x)}{\sum_{i'=1}^{N} \underline{\varphi}_{i'}(x)}\,, \qquad \underline{\varphi}_i(x) = \exp\left(-\sum_{d'=1}^{d} \frac{(x_{d'} - x_{i,d'})^2}{\sigma_{i,d'}^2}\right) \tag{15}$$

where $\underline{\varphi}_i$ are (unnormalized) Gaussian axis-oriented RBFs, x_i is the d-dimensional center of the i-th RBF, and σ_i is its d-dimensional radius. Axis-oriented RBFs were selected for a fair comparison, because the triangular fuzzy partitions are also defined separately for each variable and then combined.

Denote $\varphi(x) = [\varphi_1(x), \dots, \varphi_N(x)]^{\mathrm{T}}$. Assume that $x_1, \dots, x_N$ are all distinct from each other. Form the matrix $\boldsymbol{\varphi} = [\varphi(x_1), \dots, \varphi(x_N)] \in \mathbb{R}^{N \times N}$, which is invertible by construction. Define also a matrix $\boldsymbol{Q} \in \mathbb{R}^{N \times M}$ that collects the Q-values of the RBF centers: $\boldsymbol{Q}_{i,j} = Q(x_i, u_j)$. RBF Q-iteration uses the following approximation and projection mappings:

$$[F(\theta)](x, u_j) = \sum_{i=1}^{N} \varphi_i(x)\theta_{i,j} \ , \qquad P(Q) = (\boldsymbol{\varphi}^{-1})^{\mathrm{T}} \boldsymbol{Q} \tag{16}$$

The convergence of RBF Q-iteration to a fixed point θ^* can be guaranteed if:

$$\sum_{i'=1, i' \neq i}^{N} \varphi_{i'}(x_i) < \frac{1 - \gamma/\gamma'}{2} \tag{17}$$

for all i and some $\gamma' \in (\gamma, 1)$. Equation (17) restricts the sum of the values that the other RBFs take at the center of the i-th RBF. This is an extension of the result in [11] for normalized RBFs (the original result is given for un-normalized basis functions).

For a fair comparison with fuzzy Q-iteration, the number of RBFs is set equal to the number of fuzzy membership functions, and their centers are identical to the cores of the fuzzy membership functions. In order to have a set of radii ensuring convergence, that is a set of radii satisfying the inequalities (17), a problem involving linear constraints has been solved. The details of this procedure are left out due to space constraints.

When used with the same reward function and action quantization as fuzzy Q-iteration, RBF Q-iteration was unable to learn an appropriate solution. To help the algorithm, a finer quantization of the action space was provided: $U = \{-1, -0.2, 0, 0.2, 1\} \times \{-1, -0.2, 0, 0.2, 1\}$, and the reward function was changed to include a quadratic term in the state:

$$\rho(x, u) = -x'^{\mathrm{T}} \operatorname{diag}(0.2, 0.2, 0.1, 0.1)\, x' + \begin{cases} 10 & \text{if } \|x\|_\infty \leq 0.2 \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

where $x' = f(x, u)$ and $\operatorname{diag}(\cdot)$ denotes a square matrix with the elements given as arguments on the diagonal, and zeros elsewhere.

The results for varying damping are presented in Figure 4 (compare with Figure 3). A discrete policy was used, because it gave better results than the interpolated policy. To make the comparison with fuzzy Q-iteration easier, the original reward function (12) was used to compute the returns. Comparing the returns of the respective trajectories, the performance of the policy computed with RBF Q-iteration is worse than for fuzzy Q-iteration. From some of the initial states considered the RBF policy is not able to reach the goal region at all. In the left part of Figure 4, it can be seen that the RBF policy is incorrect on an entire vertical band around $c_x = 1$. Also, there is no sign that the high-damping regions are considered in the policy.[6]

[6] In the uniform damping case, the goal region is reached by the RBF policy from all the considered initial states, but in a longer time than using the fuzzy policy.

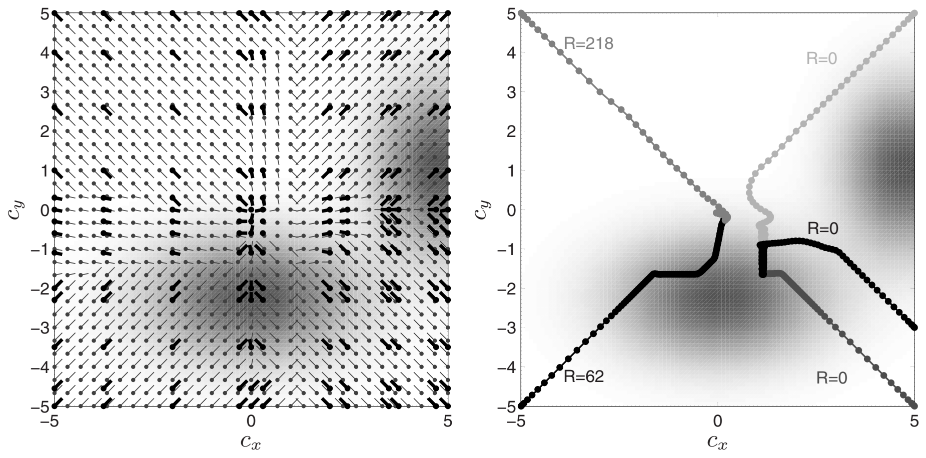

Fig. 4. *Left:* The discrete policy for RBF Q-iteration and varying damping. *Right:* A set of representative system trajectories when controlled with the RBF policy, with the associated returns.

A possible reason for the worse results of RBF approximation is that the Q-functions obtained by RBF interpolation are less smooth than those obtained with triangular fuzzy partitions (which lead essentially to multi-linear interpolation). This effect becomes more pronounced due to the convergence constraints (17) imposed on the RBF radii.

7 Possible Extensions

Although the results above were given for deterministic MDPs, fuzzy Q-iteration can be extended to stochastic problems. For instance, the asynchronous update in line 5 of Algorithm 2 becomes in the stochastic case $\theta_{i,j} \leftarrow \mathrm{E}\{\rho(x_i, u_j, x') + \gamma \max_{\underline{j}} \sum_{\underline{i}=1}^{N} \mu_{\underline{i}}(x')\theta_{\underline{i},\underline{j}}\}$. Here, x' is sampled from the density function $f(x_i, u_j, \cdot)$ of the next state given x_i and u_j. If this expectation can be computed exactly (e.g., there is a finite number of possible successor states), our results apply. In general, Monte-Carlo estimation can be used to compute the expectation. Asymptotically, as the number of samples grows to infinity, the estimate converges to the true expectation and our results can again be applied. When the number of samples is finite, [12] provides error bounds for the value iteration algorithm. These bounds could be extended to the Q-value iteration algorithm.

Fuzzy approximation can also be used online and without a model, with an arbitrary (but fixed) exploration policy, by applying the results of [17]. The same paper provides an adaptive multi-stage algorithm that converges to the true optimum as the basis functions (fuzzy membership functions, in this case) become infinitely dense in the state space.

An intermediate step is to use an offline algorithm, but with arbitrary state samples that might be different from the fuzzy cores. For this case, the dynamics and reward function of a discrete MDP with the state space equal to the set of fuzzy cores can be computed as in [19]. A solution to this discrete MDP can be computed with a model-based algorithm, and from this an approximate solution to the original problem can be derived.

8 Conclusion and Future Work

In this work, we have considered a model-based reinforcement learning approach employing parametric fuzzy approximators to represent the state-action value functions. We have proposed two different ways for updating the parameters of the fuzzy approximator, a synchronous and an asynchronous one. We have shown that both updates lead to convergent algorithms, with the asynchronous version converging at least as fast as the synchronous one. The algorithms performed well in a nonlinear control problem with four continuous state variables. Fuzzy Q-iteration also performed better than Q-iteration with normalized RBF approximation.

There exist other approximators than fuzzy partitions and RBF that could be combined with Q-iteration to yield convergent algorithms. These approximators are usually restricted classes of linear basis functions, satisfying conditions related to (but different from) Assumption 2 of Section 4. It would certainly be interesting to investigate which convergent approximator provides the best performance when combined with approximate Q-iteration.

The fuzzy approximator plays a crucial role in our approach. It determines the computational complexity of fuzzy Q-iteration, as well as the accuracy of the solution. While we considered in this paper that the membership functions were given *a priori*, we suggest as a future research direction to develop techniques to determine for a given accuracy an approximator with a small number of membership functions. The computational cost of these techniques should not be larger than using a more complex, pre-designed approximator that ensures the same accuracy. Another interesting direction would be to study the consistency properties of fuzzy Q-iteration: whether the algorithm converges to the optimal solution as the distance between fuzzy cores decreases to 0.

Acknowledgement. This research is financially supported by the BSIK-ICIS project "Interactive Collaborative Information Systems" (grant no. BSIK03024), by the NWO Van Gogh grant VGP 79-99, and by the STW-VIDI project "Multi-Agent Control of Large-Scale Hybrid Systems" (DWV.6188).

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
2. Bertsekas, D.P.: Dynamic Programming and Optimal Control, 2nd edn., vol. 2. Athena Scientific (2001)
3. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning 8, 279–292 (1992)

4. Glorennec, P.Y.: Reinforcement learning: An overview. In: ESIT 2000. Proceedings European Symposium on Intelligent Techniques, Aachen, Germany, September 14–15, 2000, pp. 17–35 (2000)
5. Horiuchi, T., Fujino, A., Katai, O., Sawaragi, T.: Fuzzy interpolation-based Q-learning with continuous states and actions. In: FUZZ-IEEE 1996. Proceedings 5th IEEE International Conference on Fuzzy Systems, New Orleans, US, September 8–11, 1996, pp. 594–600 (1996)
6. Jouffe, L.: Fuzzy inference system learning by reinforcement methods. IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews 28(3), 338–355 (1998)
7. Berenji, H.R., Khedkar, P.: Learning and tuning fuzzy logic controllers through reinforcements. IEEE Transactions on Neural Networks 3(5), 724–740 (1992)
8. Berenji, H.R., Vengerov, D.: A convergent actor-critic-based FRL algorithm with application to power management of wireless transmitters. IEEE Transactions on Fuzzy Systems 11(4), 478–485 (2003)
9. Vengerov, D., Bambos, N., Berenji, H.R.: A fuzzy reinforcement learning approach to power control in wireless transmitters. IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 35(4), 768–778 (2005)
10. Lin, C.K.: A reinforcement learning adaptive fuzzy controller for robots. Fuzzy Sets and Systems 137, 339–352 (2003)
11. Tsitsiklis, J.N., Van Roy, B.: Feature-based methods for large scale dynamic programming. Machine Learning 22(1–3), 59–94 (1996)
12. Szepesvári, C., Munos, R.: Finite time bounds for sampling based fitted value iteration. In: ICML 2005. Proceedings Twenty-Second International Conference on Machine Learning, Bonn, Germany, August 7–11, 2005, pp. 880–887 (2005)
13. Gordon, G.: Stable function approximation in dynamic programming. In: ICML 1995. Proceedings Twelfth International Conference on Machine Learning, Tahoe City, US, July 9–12, 1995, pp. 261–268 (1995)
14. Wiering, M.: Convergence and divergence in standard and averaging reinforcement learning. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 477–488. Springer, Heidelberg (2004)
15. Ormoneit, D., Sen, S.: Kernel-based reinforcement learning. Machine Learning 49(2–3), 161–178 (2002)
16. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research 6, 503–556 (2005)
17. Szepesvári, C., Smart, W.D.: Interpolation-based Q-learning. In: ICML 2004. Proceedings Twenty-First International Conference on Machine Learning, Bannf, Canada, July 4–8, 2004 (2004)
18. Singh, S.P., Jaakkola, T., Jordan, M.I.: Reinforcement learning with soft state aggregation. In: NIPS 1994. Advances in Neural Information Processing Systems 7, Denver, US, pp. 361–368 (1994)
19. Ernst, D.: Near Optimal Closed-loop Control. Application to Electric Power Systems. PhD thesis, University of Liège, Belgium (March 2003)
20. Munos, R., Moore, A.: Variable-resolution discretization in optimal control. Machine Learning 49(2–3), 291–323 (2002)
21. Sherstov, A., Stone, P.: Function approximation via tile coding: Automating parameter choice. In: Zucker, J.-D., Saitta, L. (eds.) SARA 2005. LNCS (LNAI), vol. 3607, pp. 194–205. Springer, Heidelberg (2005)