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. Well-understood RL algorithms with good convergence and consistency properties exist. 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 structure similar to those previously used for Q-learning, but we combine it with the model-based Q-value iteration algorithm. We show that the resulting algorithm converges. We also give a modified, serial 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 where pre-programmed solutions are difficult or impossible to design. Reinforcement learning (RL) is a popular learning paradigm for adaptive agents, thanks to the mildness of its assumptions on the environment (which can be a nonlinear, stochastic process), and to its ability to work without an explicit model of the environment [1–3]. At each time step, an RL agent perceives the complete state of the environment and takes an action. This action causes the environment to transit 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. Well-understood algorithms with good convergence and consistency properties are available for solving RL problems [1, 3, 4]. 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-action pair. For problems with discrete but large state-action spaces, or continuous state-action spaces, one has to rely on approximate algorithms.

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. for problems for which the state space contains an infinite (or extremely large) number of elements and the action space is discrete and not too large. 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 small number of representative actions. A significant number of (mainly model-free) fuzzy RL algorithms have been proposed, e.g., for Q-learning [5–8] or actor-critic techniques [8–13]. However, most of these algorithms are heuristic in nature, and their theoretical properties have not been investigated. Notable exceptions are the actor-critic algorithms in [10, 11].

On the other hand, a rich body of literature concerns the theoretical analysis of approximate RL algorithms, both in a model-based setting [14–17] and when an *a priori* model is not available [18–23].[3] While convergence is not ensured for an arbitrary approximator (examples of divergence have been published [14, 17, 24]), 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 [18, 19], averagers [16], interpolative representations [21]. Some authors also investigate approximators that alter their structure during learning in order to better represent the solution [19,25,26]. While some of these algorithms exhibit impressing learning capabilities, they may face convergence problems, especially if combined with non-suitable approximation structures [19, 24].

We consider here a fuzzy rule-base approximator that is similar to others previously used in fuzzy Q-learning, but we combine it with the model-based Q-value iteration algorithm. We show that the resulting algorithm is convergent. Afterwards, we propose a variant of this algorithm, which we name *serial fuzzy Q-iteration*, and which we show converges at least as fast as the original version. While serial Q-iteration has been widely used in exact RL, 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 reminds some classical 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 approximation structure. The properties of parallel and serial approximate Q-iteration using this structure are analyzed in Section 5. Section 6 illustrates the introduced algorithms on a two-dimensional navigation example. Section 7 outlines ideas for future work and concludes the paper.

2 Reinforcement Learning

In this section, we briefly introduce the RL task and characterize its optimal solution. The presentation is based on [1–3].

[3] 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.

Consider a deterministic *Markov decision process* with the state space X, the action space U, the transition function $f : X \times U \to X$, and the reward function $\rho : X \times U \to \mathbb{R}$. 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. [4,5]

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 $\gamma \in [0, 1)$. 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 $x_{k+1} = f(x_k, h(x_k))$ $\forall k$. 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 then optimal, i.e., it maximizes the return (1).

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

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

This equation states that the optimal value of action u taken in state x is the expected immediate reward plus the discounted optimal value attainable from the next state.

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

[5] Throughout the paper, 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 from the instantaneous rewards r and the return R. We denote policies by h.

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 (see e.g., [27]).

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

The Q-value iteration (or in short Q-iteration) algorithm starts from an arbitrary Q-function Q_0 and at each iteration τ updates the Q-function using the formula $Q_{\tau+1} = T(Q_\tau)$. 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 $\tau \to \infty$.

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 work without any explicit model, by learning directly the optimal Q-function from real or simulated experience in the environment (e.g., Q-learning [4]). Model-learning algorithms estimate a model from experience and use it to derive Q^* (e.g., Dyna [28]).

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 becomes 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 [21]).

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 parameter vector θ_0 and at each iteration τ updates it using the composition of the mappings P, T, and F:

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

Unfortunately, the approximate Q-iteration is not guaranteed to converge for an arbitrary approximator structure. Counter-examples can be found e.g., in [14, 24] for the related value-iteration algorithm, but those results apply directly to Q-iteration as well. One particular case in which approximate Q-iteration converges is when the composite mapping PTF can be shown to be a contraction [14, 16]. 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 scheme similar to those previously used in combination with Q-learning [5, 6, 8], 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, denoted by $U_0 = \{u_j | j = 1, \ldots, M\}$. 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 architecture 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 is 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, i.e., for every i there exists an 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 value x_i is called the center value of 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 [14].

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 state-action pair (x_i, u_j).[6] The approximator takes as input the state-action pair (x, u_j) and outputs the Q-value:

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

[6] 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 $\|\theta\|_\infty = \max_{i,j} |\theta_{i,j}|$.

This is a 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 approximator (7) is equivalent to a Takagi-Sugeno rule-base with one input, the state x, and M singleton outputs, the Q-values corresponding to each of the discrete actions $u_1, \ldots, u_M$. The i-th rule in this rule-base has the form:

$$R_i: \quad \textbf{if } x \textbf{ is } \mathcal{X}_i \textbf{ then } Q(x,u_1) = \theta_{i,1} \textbf{ and } Q(x,u_2) = \theta_{i,2} \ldots \textbf{ and } Q(x,u_M) = \theta_{i,M}$$

The logical expression 'x **is** $\mathcal{X}_i$' is here characterized true with degree $\mu_i(x)$, the membership degree of x in $\mathcal{X}_i$. The fuzzy rule-base outputs the weighted sum of the consequent values $\theta_{i,j}$ in each rule, where the weight factor of a particular rule i corresponds to the degree of fulfillment of its logical expression. This weighted sum is written as (7).

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

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

The approximation structure (7), (8) shares some strong similarities with several classes of approximators that have already been used in RL: interpolative representations [14], averagers [16], and representative-state techniques [23].

The Q-iteration algorithm using the approximation architecture (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 $[T(\widehat{Q}_\tau)](x_i, u_j)$, where $\widehat{Q}_\tau = F(\theta_\tau)$. Hence, line 4 can be written $\theta_{\tau+1,i,j} \leftarrow [PTF(\theta_\tau)]_{i,j}$ 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 serial fashion, this version is called serial Q-iteration. Although the exact counterpart of this algorithm is widely used [1, 3], the serial variant has receive little attention in the context of approximate RL. To differentiate between the two versions, Algorithm 1 is hereafter called parallel fuzzy Q-iteration.

5 Convergence of Fuzzy Q-iteration

In this section, the convergence of parallel fuzzy Q-iteration and serial fuzzy Q-iteration is established, i.e., it is shown that there exists a parameter vector θ^* such that for both algorithms, $\theta_\tau \to \theta^*$ as $\tau \to \infty$. In addition, serial fuzzy Q-iteration is shown to converge at least as fast as the parallel 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 [14, 16]. The consistency of the fuzzy Q-iteration, i.e., the convergence to the optimal Q-function Q^* as

Algorithm 1 Parallel fuzzy Q-iteration

$\theta_0 \leftarrow 0$; $\tau \leftarrow 0$
repeat
  for $i = 1, \ldots, N, j = 1, \ldots, M$ **do**
    $\theta_{\tau+1,i,j} \leftarrow \rho(x_i, u_j) + \gamma \max_{\underline{j}} \sum_{\underline{i}=1}^{N} \mu_{\underline{i}}(f(x_i, u_j))\theta_{\tau,\underline{i},\underline{j}}$
  end for
  $\tau \leftarrow \tau + 1$
until $\|\theta_\tau - \theta_{\tau-1}\|_\infty \leq \delta$

Algorithm 2 Serial fuzzy Q-iteration

$\theta_0 \leftarrow 0$; $\tau \leftarrow 0$
repeat
  $\theta \leftarrow \theta_\tau$
  for $i = 1, \ldots, N, j = 1, \ldots, M$ **do**
    $\theta_{i,j} \leftarrow \rho(x_i, u_j) + \gamma \max_{\underline{j}} \sum_{\underline{i}=1}^{N} \mu_{\underline{i}}(f(x_i, u_j))\theta_{\underline{i},\underline{j}}$
  end for
  $\theta_{\tau+1} \leftarrow \theta$
  $\tau \leftarrow \tau + 1$
until $\|\theta_\tau - \theta_{\tau-1}\|_\infty \leq \delta$

the maximum distance between the centers of adjacent fuzzy sets goes to 0, is not studied here, and is a topic for future research.

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

Proof. The proof follows from the proof of convergence of (parallel) value iteration with averagers [16], or with interpolative representations [14]. This is because fuzzy approximation is an averager by the definition in [16], and an interpolative representation by the definition in [14]. The main idea of the proof is that PTF is a contraction with factor γ, i.e., $\|PTF(\theta) - PTF(\theta')\|_\infty \leq \gamma \|\theta - \theta'\|_\infty$, for any θ, θ'. This is true thanks to the non-expansive nature of P and F, and because T is a contraction. □

Similarly to the convergence proof for exact serial value iteration in [3], it is shown below that the *approximate* serial Q-iteration Algorithm 2 converges.

Proposition 2. *Serial 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 $l = 0, \ldots, n$ recursively the mappings $S_l : \mathbb{R}^n \to \mathbb{R}^n$ as:

$$S_0(\theta) = \theta$$

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

In words, S_l corresponds to updating the first l parameters using approximate serial Q-iteration, and S_n is a complete iteration of the approximate serial 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:

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

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, serial fuzzy Q-iteration converges. □

This proof is actually more general, showing that approximate serial Q-iteration converges for any approximator structure F and projection P for which PTF is a contraction. It can also be easily shown that parallel 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 parallel or serial (element-by-element) fashion.

We now show that serial fuzzy Q-iteration converges at least as fast as the parallel 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. *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 that $Q \leq Q'$:

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

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 that $Q \leq Q'$, 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 [3]. □

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

The following bounds on the sub-optimality of the convergence point, and of the policy corresponding to this point, follow from [14, 16]. However, these bounds apply only when the action space of the *original* problem is discrete (i.e., no discretization is applied prior to fuzzy Q-iteration).

Proposition 4. *If the original action space is discrete and $\min_{\underline{Q}} \left\|Q^* - \underline{Q}\right\|_\infty = \varepsilon$ where $\underline{Q}$ is any fixed point of the composite mapping $FP : \mathcal{Q} \to \mathcal{Q}$, then serial and parallel fuzzy Q-iteration converge to θ^* such that:*

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

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

where $Q^{\widehat{h}^*}$ is the action-value 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 approximation structure,

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 parallel and serial fuzzy Q-iteration, it was shown that the serial version converges at least as fast as the parallel one, and therefore might be more desirable in practice. When the original action space is discrete, bounds on the derived Q-function and policy were also shown to hold. This provides a degree of 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) 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-varying damping coefficient (friction). All the state and action variables are bounded. The bounds are listed in Table 1, which also collects the meaning and measuring 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 discretization points.[7]

The goal of reaching the origin in minimum time is expressed by the following reward function:

$$\rho(x, u) = \begin{cases} 10 & \|x\|_\infty \leq 0.1 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

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

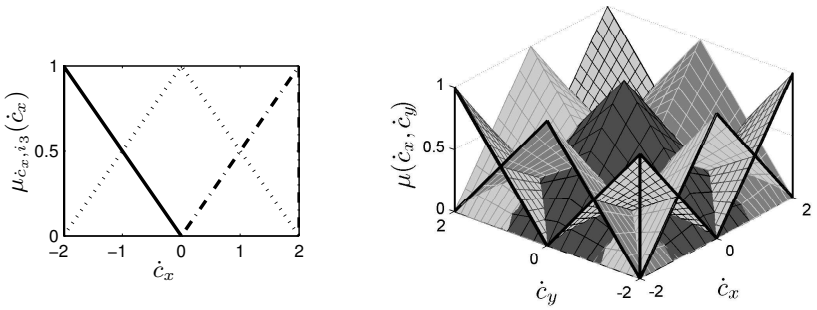

Fig. 1. *Left:* triangular fuzzy partition for $\dot{c}_x \in [-2, 2]$. Each membership function is plotted in a different line style. The partition for $\dot{c}_y$ is identical. *Right:* composition of the fuzzy partitions for $\dot{c}_x, \dot{c}_y$, yielding the two-dimensional fuzzy partition for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

This means that both of the coordinates and the velocities have to be smaller than 0.1 in magnitude for the agent to get a non-zero reward.

The control force is quantized into 9 discrete values: $\{-1, 0, 1\} \times \{-1, 0, 1\}$. 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 centers completely determines the shape of the membership functions.

For the position coordinates, triangular partitions are used as well, the centers of which vary between the two particular damping landscapes considered in the sequel. 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: $b(c_x, c_y) = b_0 = 0.5$ kg/s. Identical triangular fuzzy partitions were defined for c_x and c_y, with the centers in $\{-5, -2, -0.3, -0.1, 0, 0.1, 0.3, 2, 5\}$. Serial and parallel fuzzy Q-iteration were run with a discount factor $\gamma = 0.98$ and a threshold $\delta = 0.01$ (see Algorithms 1 and 2). The parameters γ and δ are set somewhat arbitrarily, but

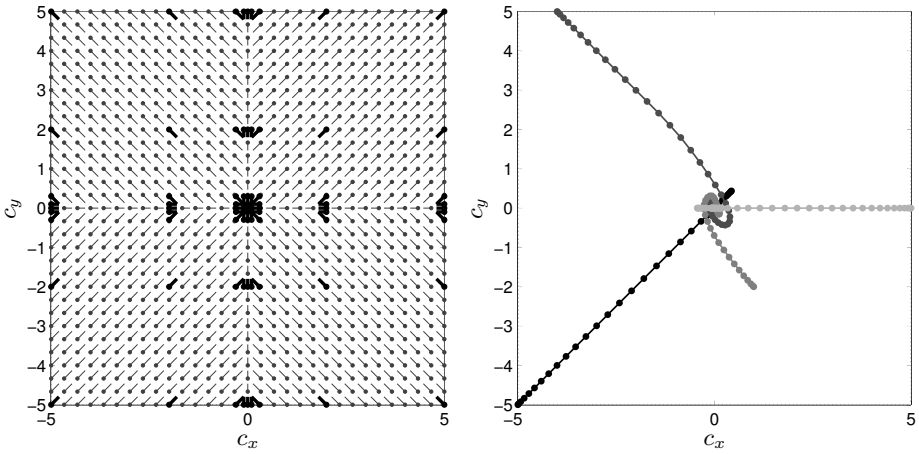

Fig. 2. *Left:* The policy for constant damping, when $\dot{c}_x = \dot{c}_y = 0$. The direction and magnitude of the control force in a grid of sample points (marked by dots) is indicated by lines. Thick, black lines indicate exact policy values in the centers of the fuzzy partition (marked by thick, 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, from which a rough impression of the speed can be obtained.

their variation around the given values does not significantly affect the computed policy. The serial algorithm converged in 339 iterations; the parallel one in 343. Therefore, in this particular problem, the speed of convergence for the serial algorithm is close to the speed for the parallel 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 center state x_i, $j_i^* = \arg\max_j [F(\theta^*)](x_i, u_j) = \arg\max_j \theta^*_{i,j}$.

Figure 2 presents a slice through the computed policy for zero velocity, $\widehat{h}^*(c_x, c_y, 0, 0)$, together with a few sample trajectories. This slice is clearly different from the optimal continuous-action policy, which would steer the agent directly towards the goal zone regardless of the position. Also, since the actions are originally continuous, the bound (10) does not apply. Nevertheless, the slice presented in the figure is close to the best that can be achieved under 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 d Gaussian functions:

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

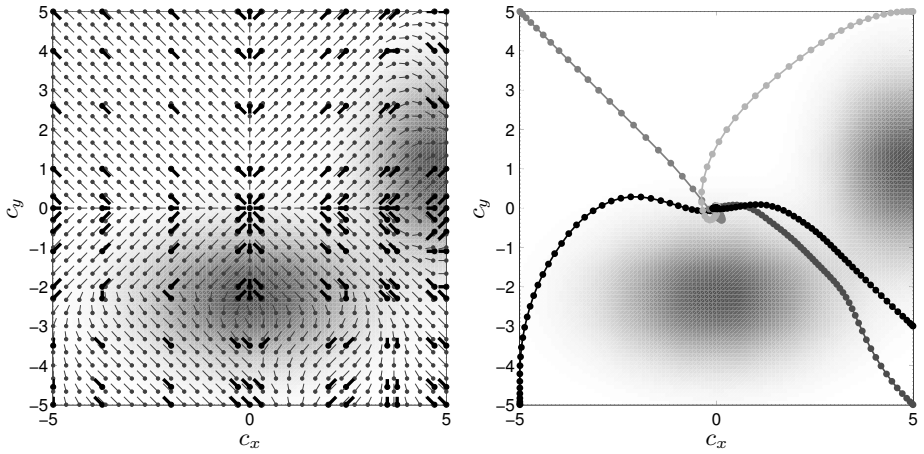

Fig. 3. *Left:* The policy for varying damping (14), when $\dot{c}_x = \dot{c}_y = 0$. Darker areas indicate larger damping. The direction and magnitude of the control force in a grid of sample points is indicated. The thick, black lines indicate exact policy values in the centers of the fuzzy partition.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

The chosen values were: $b_0 = 0.5$, $d = 2$, $b_1 = b_2 = 8$, $g_{x,1} = 0$, $g_{y,1} = -2.3$, $\sigma_{x,1} = 2.5$, $\sigma_{x,2} = 1.5$, 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 has two maxima equal to 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 centers for the position partition are marked by thick black dots in Figure 3, left. They were chosen making use of prior knowledge about the position of the high-friction areas. The centers include representative points around these areas, and some points near the goal. Serial and parallel 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 a few 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 partition centers.

7 Conclusion and Future Work

In this work, we have considered a model-based reinforcement learning approach employing parametric fuzzy architectures to represent the state-action value functions. We have proposed two different ways for updating the parameters of the fuzzy architecture, a parallel and a serial one. We have shown that in both cases the algorithms were converging, with the serial version converging at least as fast as the parallel one. The algorithms exhibited good performance in a nonlinear control problem with four continuous state variables.

The fuzzy approximation architecture 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 approximation architecture was given *a priori*, we suggest as a first future research direction to develop techniques able to determine for a given accuracy some low-complexity approximation architectures. We stress that such techniques should not imply computational burdens that undermine the benefits of using the resulting low-complexity ϵ-accurate approximator, over more complex but predesigned ϵ-accurate approximators relying e.g., on a triangulation of the state space.

A second direction for future work is the study of the consistency properties of the fuzzy Q-iteration: whether the algorithm converges to the optimal solution as the distance between fuzzy centers decreases to 0. A third direction is the search for online (model-learning or model-free) fuzzy RL algorithms which have good learning speed and low computational complexity. Finally, while most of the research in RL for designing good approximation architectures focuses on problems having simple action spaces, we think it would be interesting to extend our approach such that it can handle directly (without discretization) complex or continuous action spaces.

Acknowledgement: This research is financially supported by Senter, Dutch Ministry of Economic Affairs within 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, US (1998)
2. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research **4** (1996) 237–285
3. Bertsekas, D.P.: Dynamic Programming and Optimal Control. 2nd edn. Volume 2. Athena Scientific (2001)
4. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8** (1992) 279–292
5. Bonarini, A., Montrone, F., Restelli, M.: Reinforcement distribution in continuous state action space fuzzy Q-learning: A novel approach. In: Proceedings 6th International Workshop on Fuzzy Logic and Applications (WILF-05), Milan, Italy (15–17 September 2005)
6. Glorennec, P.Y.: Reinforcement learning: An overview. In: Proceedings European Symposium on Intelligent Techniques (ESIT-00), Aachen, Germany (14–15 September 2000) 17–35
7. Horiuchi, T., Fujino, A., Katai, O., Sawaragi, T.: Fuzzy interpolation-based Q-learning with continuous states and actions. In: Proceedings 5th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE-96), New Orleans, US (8-11 September 1996) 594–600
8. Jouffe, L.: Fuzzy inference system learning by reinforcement methods. IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews **28**(3) (1998) 338–355
9. Berenji, H.R., Khedkar, P.: Learning and tuning fuzzy logic controllers through reinforcements. IEEE Transactions on Neural Networks **3**(5) (1992) 724–740

10. 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) (2003) 478–485
11. 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) (2005) 768–778
12. Bonarini, A.: Evolutionary learning, reinforcement learning, and fuzzy rules for knowledge acquisition in agent-based systems. Proceedings of the IEEE **89**(9) (2001) 1334–1346
13. Lin, C.K.: A reinforcement learning adaptive fuzzy controller for robots. Fuzzy Sets and Systems **137** (2003) 339–352
14. Tsitsiklis, J.N., Van Roy, B.: Feature-based methods for large scale dynamic programming. Machine Learning **22**(1–3) (1996) 59–94
15. Szepesvári, C., Munos, R.: Finite time bounds for sampling based fitted value iteration. In: Proceedings Twenty-Second International Conference on Machine Learning (ICML-05), Bonn, Germany (7–11 August 2005) 880–887
16. Gordon, G.: Stable function approximation in dynamic programming. In: Proceedings Twelfth International Conference on Machine Learning (ICML-95), Tahoe City, US (9–12 July 1995) 261–268
17. Wiering, M.: Convergence and divergence in standard and averaging reinforcement learning. In: Proceedings of the 15th European Conference on Machine Learning (ECML'04), Pisa, Italy (20–24 September 2004) 477–488
18. Ormoneit, D., Sen, S.: Kernel-based reinforcement learning. Machine Learning **49** (2002) 161–178
19. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research **6** (2005) 503–556
20. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. Journal of Machine Learning Research **4** (2003) 1107–1149
21. Szepesvári, C., Smart, W.D.: Interpolation-based Q-learning. In: Proceedings 21st International Conference on Machine Learning (ICML-04), Bannf, Canada (July 4–8 2004)
22. Singh, S.P., Jaakkola, T., Jordan, M.I.: Reinforcement learning with soft state aggregation. In: Advances in Neural Information Processing Systems 7, Denver, Colorado, USA (1994) 361–368
23. Ernst, D.: Near Optimal Closed-loop Control. Application to Electric Power Systems. PhD thesis, University of Liège, Belgium (March 2003)
24. Boyan, J., Moore, A.: Generalization in reinforcement learning: Safely approximating the value function. In: Advances in Neural Information Processing Systems 7 (NIPS-94), Denver, Colorado, US (1994) 369–376
25. Munos, R., Moore, A.: Variable-resolution discretization in optimal control. Machine Learning **1** (2001) 1–31
26. Sherstov, A.A., Stone, P.: Function approximation via tile coding: Automating parameter choice. In: Proceedings 6th International Symposium on Abstraction, Reformulation and Approximation (SARA-05). Volume 3607 of Lecture Notes in Computer Science., Airth Castle, Scotland, UK (26–29 July 2005) 194–205
27. Jodogne, S.: Closed-Loop Learning of Visual Control Policies. PhD thesis, University of Liège, Belgium (December 2006)
28. Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings Seventh International Conference on Machine Learning (ICML-90), Austin, Texas, US (June 21–23 1990) 216–224