Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)

Q-LEARNING WITH CONTINUOUS STATE SPACES AND FINITE DECISION SET

KENGY BARTY, PIERRE GIRARDEAU, JEAN-SÉBASTIEN ROY AND CYRILLE STRUGAREK

EDF R&D
1, avenue du Général de Gaulle
92141 Clamart Cedex, France.

Abstract. **This paper aims to present an original technique in order to compute the optimal policy of a Markov Decision Problem with continuous state space and discrete decision variables. We propose an extension of the Q-learning algorithm introduced in 1989 by Watkins for discrete Markov Decision Problems. Our algorithm relies on stochastic approximation and functional estimation, and uses kernels to locally update the Q-functions. We state under mild assumptions a converge theorem for this algorithm. Finally, we illustrate our algorithm by solving two classical problems: the Mountain car Task and the Puddle World Task.**

I. Introduction

In a Markov Decision Problem (MDP), an agent wanders in a Markovian environment and tries to minimize its expected long-term reward (or to minimize its long-term cost), by performing actions that only have to depend on the current state.

Simple examples of MDPs are concerned with leading an agent moving on a surface to a certain goal in shortest time, when its trajectory may be affected by some sort of deterministic or stochastic process (wind for example). More complicated tasks may be written using the mathematical model of MDPs, such as controlling an hydro-power plant that has to satisfy a demand over a certain period of time, while minimizing the cost of the thermal power production if the hydro-power plant cannot supply the demand completely.

Dynamic programming is a powerful methodology for dealing with sequential decision making problems under uncertainty like MDPs. In the case of a continuous state space, the usual approach is to discretize the state space and to apply recursively the Bellman operator. This discretization usually leads to very large state spaces, and is known as the curse of dimensionality. An additional complexity arises in the stochastic case, since the conditional expectation appearing in the Bellman equation must also be approximated through a discretization of the dynamics.

However, in the MDP setting, reinforcement learning combined with the theory of dynamic programming led to very efficient algorithms in the case of a discrete state space, via the TD(λ) algorithm of Sutton [15] and the Q-learning algorithm of Watkins [18]. Moreover, it is proved that Q-learning (cf. [18, 19, 9]) and TD(λ) (cf. [17, 8]) algorithms converge with probability one.

Unfortunately, in the case where the state space is continuous, discretizing can only lead to near-optimal solutions. Ormoneit and Sen [12] or Glynn [11] recently proposed to estimate the value functions using non-parametric regression methods, such as kernel-based methods. They showed that their algorithm could be applied even when classical algorithms based on discretization of the state space failed to converge. A major drawback is that the method is not recursive: it approximates the value function using estimation points, and when one wants to increase the number of estimation points, the previous estimate cannot be used to derive the new one.

We present an algorithm that extends Q-learning to the case of a continuous state space, by using local updates with kernels to estimate the value functions. Our method is recursive and non-parametric. The characteristic of a recursive method is the possibility to increase the accuracy of the approximation by a simple update of the previous value. It is based on stochastic approximation (see [13], or [10] for an historical survey of these techniques). Since we avoid the space discretization, our method leads to the optimal solution of the original problem. Moreover, it is convenient from a practical point of view since it avoids discretizing the dynamics.

E-mail address: jean-sebastien.roy@edf.fr, kengy.barty@edf.fr.

Ormoneit and Sen's algorithm [12] differs from our since it is not recursive, therefore it requires to set a priori finite collection of kernels and then iteratively apply their estimate of the Bellman operator in order to obtain an approximation. Their main result ensures that approximation converges in probability to the true Bellman function as the cardinal of kernels increases. On the contrary we do not compute directly from a set of historical outcomes but we improve our estimate of the Bellman function as new data are observed.

In section II, we present the Q-learning formulation. Afterward we introduce the Q-learning algorithm, which is closely related to our proposed method. In the same section, we present our kernel method and state a theorem that summarizes the properties of our algorithm. Then, in section IV, we solve the mountain car task, compare with Jong's results [9] and show how the proposed algorithm is efficient in this case. We also address the Puddle World Task which involve random variables.

II. Theoretical Framework

II.1. **The Q-learning formulation.** Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, all random variables are supposed to be defined from Ω onto another measurable space. If G is a random variable, its probability distribution is denoted $\mathbb{P}_G$. Let us consider an optimal control problem with a finite set of decision and dates $\{0, 1, \ldots, T\}$. The uncertainties W are supposed to be independent random variables $(W_j)_{1\le j\le T}$ with values in $\mathbb{R}^m$. Let $(X_j)_{1\le j\le T}$ be the controlled random process with values in $S \subseteq \mathbb{R}^d$ defined by:

$$X_{j+1} = f_j\left(X_j, \gamma_j\left(X_j\right), W_{j+1}\right), \forall j \in \{0, \ldots, T-1\},$$

with: $\gamma_j : S \to \mathbb{U}_j^{ad}$ the decision rule.

We denote by $L_j\left(X_j, \gamma_j\left(X_j\right), W_{j+1}\right)$ the cost at state X_j, when action $\gamma_j\left(X_j\right)$ is taken, and uncertainty W_{j+1} occurs. We also consider a cost at final time denoted $G\left(X_T\right)$. Our aim is to minimize the expected global cost, starting from a state $X_0 \in S$, i.e.:

(1)
$$V_0^*\left(X_0\right) = \min_{(\gamma_j)} \mathbb{E}\left[\sum_{j=0}^{T-1} L_j\left(X_j, \gamma_j\left(X_j\right), W_{j+1}\right) + G\left(X_T\right) \middle| X_0\right], \quad \text{with } \gamma_j : S \to \mathbb{U}_j^{ad}, \quad \forall j = 0, \ldots, T.$$

Now let us note $V_j^*(x)$ the expectation of the future cost under the optimal policy, starting from state x at time j. Equation (1) can be rewritten as a dynamic programming equation, let $x \in S$:

(2)
$$\begin{cases} V_j^*(x) = \min_{u\in\mathbb{U}_j^{ad}} \mathbb{E}\left[L_j\left(x, u, W_{j+1}\right) + V_{j+1}^*\left(f_j\left(x, u, W_{j+1}\right)\right)\right], \\ V_N^*(x) = G(x), \end{cases}$$

We now present the Q-learning counterpart of equation (2). Let us denote by $Q_j^*(x, u)$ the expectation of the future cost starting from state x and taking action u at time j. We have the following relation between Q_j^* and V_{j+1}^* valid for $j = 1, \ldots, T-1$:

$$Q_j^*(x, u) = \mathbb{E}\left[L_j\left(x, u, W_{j+1}\right) + V_{j+1}^*(f_j(x, u, W_{j+1}))\right].$$

Moreover, Q_j^* can be derived from Q_{j+1}^* as follows:

$$\begin{aligned} Q_j^*(x,u) &= \mathbb{E}\left[L_j\left(x, u, W_{j+1}\right) + \min_{v\in\mathbb{U}_{j+1}^{ad}} Q_{j+1}^*\left(f_j\left(x, u, W_{j+1}\right), v\right)\right], \\ Q_N^*(x,u) &= G(x), \forall x \in S. \end{aligned}$$

Preceding equations can be seen as a fixed point equation on $Q^* = (Q_t^*)_{0\le t\le T}$.

II.2. **The Q-learning algorithm.** We now consider the feasible decision sets $\left(\mathbb{U}_j^{ad}\right)_{j=0,\ldots,T}$ to be finite. At this step, the policy γ_j at each time j is still a mapping from the state space S onto the feasible decision set $\mathbb{U}_j^{ad}$. The classical Q-learning approach [16, 19, 17, 8] applies in the case where the state space S is discrete and not too large.

In order to estimate $Q_j^*(x,u)$, the Q-learning algorithm uses the following update rule:

$$Q_j^{k+1}(x,u) = Q_j^k(x,u) + \rho^k \Delta_j^{k+1}(x,u), \quad \forall (x,u) \in S \times \mathbb{U}_j^{ad},$$

$$\Delta_j^{k+1}(x,u) = L_j\left(x, u, W_{j+1}^{k+1}\right) + \min_{v\in\mathbb{U}_{j+1}^{ad}} Q_{j+1}^k\left(f_j\left(x, u, W_{j+1}^{k+1}\right), v\right) - Q_j^k(x,u),$$

and W_j^k is a realization of the process W_j. Note that at each iteration the update is performed for every state and action (x,u), and each time step.

Instead of updating Q_j^k for every state and action (x,u), Sutton [15] proposed to randomize this operation by drawing realizations X_j^k of the random variable X_j.

II.3. **Q-learning with kernels.** Let $(\mathbb{U}_j^{ad}, \mathcal{P}(\mathbb{U}_j^{ad}), \pi^j)$ be a probability space. Henceforth we will suppose we have always $(\mathbb{U}_j^{ad}, \mathcal{P}(\mathbb{U}_j^{ad}), \pi^j) = (\mathbb{U}^{ad}, \mathcal{P}(\mathbb{U}^{ad}), \pi)$. We propose an alternative approach that is non-parametric and avoids any a priori discretization of the state space. However, decision space $\mathbb{U}^{ad}$ is still assumed discrete. The finite decision space $\mathbb{U}^{ad}$ is randomly explored along the iterations as well by drawing possible decisions u according to probability π.

Our algorithm reads as follows:

Algorithm II.1. *Initialize Q_j^0 to 0 for all $j \in \{0, \ldots, T-1\}$, Step $k \ge 0$:*

- *Draw X_0^k according to X_0 independently from the past samples,*
- *draw $\left(W_j^k\right)_{1\le j\le T}$ according to the probability distribution of the random variable $(W_1, \ldots, W_T)$ independently from the past samples,*
- *draw $U^k = \left(U_j^k\right)_{0\le j\le T-1}$ according to the probability distribution $\pi^{\otimes T}$ independently from the past samples,*
- *finally compute $X^k = \left(X_j^k\right)_{1\le j\le T}$ according to:*

$$X_{j+1}^k = f_j\left(X_j^k, U_j^k, W_{j+1}^k\right).$$

- *Update the value functions Q_j^k:*

$$\left\{\begin{array}{lcl} Q_T^k(x,u) & = & G(x), \quad \forall x,u, \\ & \vdots & \\ Q_j^k(\cdot,\cdot) & = & Q_j^{k-1}(\cdot,\cdot)+\rho^{k-1}\,\Delta_j^k\,K_j^{k-1}(X_j^k,U_j^k,\cdot,\cdot), \\ & \vdots & \\ Q_0^k(\cdot,\cdot) & = & Q_0^{k-1}(\cdot,\cdot)+\rho^{k-1}\,\Delta_0^k\,K_0^{k-1}(X_0^k,U_0^k,\cdot,\cdot), \end{array}\right.$$

where, for all $j=0,\dots,T-1, \Delta_j^k=\Delta_j^k(X_j^k,U_j^k)$.

Functions K_j^k are kernels, a typical choice for these mappings is the Gaussian kernel:

$$K_j^k(x,u,x',u') = \left\{\begin{array}{l} e^{-\left\|\frac{x-x'}{\eta^k}\right\|^2} \text{ if } u=u', \\ 0 \text{ otherwise.} \end{array}\right. \tag{3}$$

where $\eta^k \to 0$ when $k \to +\infty$.

Since we draw the decision u independently from the number of iterations k, X_j follows a law of probability that is independent of k. Thus, we can define the following inner products and norms where $\mu_j = \mathbb{P}_{X_j} \otimes \pi$ and $\mu_j = \mathbb{P}_{W_j} \otimes \mathbb{P}_{X_j} \otimes \pi$:

$$\langle g,h\rangle_{\mu_j} = \mathbb{E}\left[\sum_{i=1}^{card(\mathbb{U}_j^{ad})} \pi_{u_i} g\left(X_j,u_i\right) h\left(X_j,u_i\right)\right],$$

$$\|e\|_{\nu_j}^2 = \mathbb{E}\left[\left\langle e\left(f_{j-1}\left(\cdot,\cdot,W_j\right)\right), e\left(f_{j-1}\left(\cdot,\cdot,W_j\right)\right)\right\rangle_{\mu_{j-1}}\right],$$

Moreover, we introduce:

$$v_j^k\left(x\right) \in \arg\min_{v\in\mathbb{U}^{ad}} Q_j^k\left(x,v\right),$$

and:

$$r_j^k\left(x,u\right) = \mathbb{E}^k\left[\Delta_j^{k+1}(x,u)\right].$$

Finally, we denote by $V_j^k\left(x\right) = \min_{u\in\mathbb{U}^{ad}} Q_j^k\left(x,u\right)$ the k-th approximation of the Bellman function on x.

Theorem II.1. *If $\min_{u\in\mathbb{U}^{ad}} \pi(u) > 0$ and if for all $j \in \{0,\dots,T\}$, there exists $b_j \in \mathbb{R}$ such that:*

(4)
$$\left\|r_j^k\left(\cdot,\cdot\right) - \mathbb{E}^k\left[r_j^k(X_j^{k+1},U_j^{k+1})\frac{1}{\epsilon^k}K^k\left(X_j^{k+1},U_j^{k+1},\cdot,\cdot\right)\right]\right\|_{\mu_j} \leq b_j\epsilon^k(1+\|r_j^k\|_{\mu_j}),$$

$$\mathbb{E}^k\left[\left\|K^k\left(X_j^{k+1},U_j^{k+1},\cdot,\cdot\right)\right\|_{\mu_j}^2\right] < \epsilon^k, \tag{5}$$

$$\sum_{k\in\mathbb{N}}(\rho^k)^2\epsilon^k < \infty, \quad \sum_{k\in\mathbb{N}}\rho^k(\epsilon^k)^2 < +\infty, \quad \sum_{k\in\mathbb{N}}\rho^k\epsilon^k = +\infty, \tag{6}$$

where $\epsilon^k = (\eta^k)^d$ and η^k is the bandwidth of the kernel. Then Q-functions Q_j^k defined by Algorithm II.1 converge a.s., when $k \to +\infty$, to the solution.

The proof of the theorem II.1 can be found in [2] and follows from the general result published in [4]. Indeed, the algorithm II.1 is a perturbed sub-gradient algorithm and can be written as follows:

$$Q^{k+1} = Q^k + \rho^k\epsilon^k(s^k + w^k),$$

where $s_j^k = r_j^k$ and w^k is defined by:

$$w_j^k = \Delta_j^k\frac{1}{\epsilon^k}K^k(X_j^k,U_j^k,\cdot,\cdot) - r_j^k.$$

The vector s^k is a descent direction for the Lyapunov function $\mathcal{L}(Q) = \frac{1}{2}\sum_{j=1}^T \|Q_j - Q_j^*\|_{\mu_j}^2$. We then need that the perturbation w^k satisfies some statistical assumptions in order to guarantee the convergence of the algorithm. When the state space is finite the perturbation is generally assumed to be a martingale difference, so its conditional expectation with respect to the past perturbations is equal to zero. Since we suppose that the state space could be continuous we can only make the assumption that w^k is asymptotically a martingale difference. This hypothesis is translated into the theorem II.1 by assumption (4). Assumption (5) on the second order moment of the kernel function prevent the perturbation w^k to disturb to much the descent direction estimation. The stepsize relations (6) are quite classical in stochastic optimization. Remember that ϵ^k is directly linked with the bandwidth of the kernel, and ρ^k is the size of the step performed by the algorithm in the estimated descent direction. The stepsize relations express that:

- the sequence $(\rho^k)_{k\in\mathbb{N}}$ locally satisfies the properties required for classical open-loop stochastic gradient algorithm [5],
- the sequence $(\epsilon^k)_{k\in\mathbb{N}}$ converges to zero, so that our kernels approximation converges to the true Bellman function.

We have considered a finite horizon problem therefore we don't need to rely on the existence of an invariant probability distribution of the state variable X in order to prove either the existence of a solution for the dynamic equation or to implement our algorithm. Nevertheless, under similar assumptions, our methodology can also be applied to infinite horizon or free terminal time problems. Concerning the drawings of the different random variables the only restriction is to compute or to observe X_{j+1}^k according to its conditional probability distribution knowing state X_j^k and action U_j^k.

III. Practical discussion and implementation details

In this section, we discuss the techniques useful for a practical implementation of our algorithm.

III.1. **Numerical complexity.** At each step of algorithm II.1, the Q_j^k functions are weighted sums of kernels, and therefore are completely characterized by the centers (X_j^{k+1},U_j^{k+1}), the bandwidths, and the weights $\rho_j^k\Delta_j^{k+1}$ of the kernels, i.e.:

$$Q_j^{k+1}(\cdot,\cdot) = \sum_{l=0}^k \rho^l\Delta_j^{l+1}K_j^l(X_j^{l+1},U_j^{l+1},\cdot,\cdot).$$

The numerical complexity of the algorithm is therefore determined by the number of operations required to evaluate and update the functions Q_j^k.

While the algorithm can be easily implemented by storing the required coefficients as a list, the computational complexity would be quadratic in the number of iterations, since the number of kernels grows linearly with the number of iterations. In practice, the use of the Fast Gauss Transform techniques of [7], enables us to compute and update the Q_j^k functions in almost constant time, so that the numerical complexity of the algorithm is effectively linear in the number of iterations. The amount of storage required by Fast Gauss Transform is also only a function of the required precision. Nevertheless, even recent improvements of these techniques (see, e.g, [20]) do not scale well with the dimension, so that the naive implementation might be the most efficient for a limited amounts of iterations, and a number of dimensions greater than, say, 5.

Remark also that the complexity is difficult to compare in itself with other, non-recursive learning algorithms. Contrarily to most regression based learning algorithms, the coefficients of the kernel sum are not globally optimized: each kernel is added without updating the weights of the previous kernels.

III.2. **On-policy control drawings.** For the efficient use of this algorithm, a question remains open: how to draw policies efficiently ? The only condition we need for convergence is that every possible policy shall be selected infinitely often. In the convergence proof this condition is written as $\min_{u\in\mathbb{U}^{ad}} \pi(u) > 0$. There are two classical approaches to ensure this, which are often called *on-policy* methods and *off-policy* methods.

Off-policy methods do not take into account the growth of our knowledge of the Q-functions along the iterations. They typically consist in choosing a priori distribution of the policies to be tested all along the iterations.

On the contrary, on-policy methods aim at selecting more and more policies that seem relevant according to our knowledge of the Q-functions at the current iteration. However, to ensure convergence, we shall still sometimes test policies at random. This is what practitioners call *soft* on-policy control methods.

We choose to test policies in a α-greedy way, which is an example of soft on-policy method [16, section 5.4]. In most cases (with probability $1-\alpha$), we choose the optimal policy according to our estimate of the Q-functions at the current step, i.e. we choose $u \in \arg\min_{v\in\mathbb{U}_j^{ad}} Q_j^k(x,v)$. However, to ensure the convergence of the algorithm, we draw random policies with probability α.

This technique allows the algorithm to explore the areas where policies seem to be optimal more often.

III.3. **On-policy step-sizes.** Another way to improve the computational abilities of our approach is to develop adequate procedures to fit the stepsizes. It is worth noting that stochastic algorithms are very sensitive to their choice. In [3], the authors propose a heuristic to fit, at iteration $k+1$, the steps ϵ^k and ρ_j^k in as a function of the current state X_j^{k+1} and decision U_j^{k+1}. The idea is the following. The Q_j function will be updated around (X_j^{k+1}, U_j^{k+1}), in a neighborhood defined by ϵ^k, and with an intensity ρ_j^k. The next time we fall in this neighborhood, we may want to have a new neighborhood and a new intensity only slightly lower than the preceding ones, since the samples between those two steps did not contribute to the Q_j function in this neighborhood. We hence propose an adaptive way to fit the stepsizes to the state-decision pair. Let us use sequences defined separately for each $j\in\{0,\dots,T\}$, i.e., (ϵ_j^k) and (ρ_j^k). ϵ_j^k and ρ_j^k will be used to update the Q_j function. Using these sequences, let us define iteratively for all $k\in\mathbb{N}$ and all $j\in\{0,\dots,T\}$, the mappings $f_j^k : S\times\mathbb{U}_j^{ad}\to\mathbb{R}$ by:

$$f_j^k(\cdot,\cdot) = \sum_{l=0}^{k-1} \frac{1}{\epsilon_j^l} K_j^l(X_j^{l+1}, U_j^{l+1}, \cdot, \cdot).$$

Hence, for all $k\in\mathbb{N}$ and all $j\in\{0,\dots,T\}$, f_j^k/k can be considered as an approximation of the density function of the drawings (X_j, U_j). Let us now define for all $k\in\mathbb{N}$ and all $j\in\{0,\dots,T\}$, $g_j^k = \lfloor f_j^k(X_j^{k+1}, U_j^{k+1})\rfloor$. The larger is g_j^k, the more the neighborhood of (X_j^{k+1}, U_j^{k+1}) has been explored in the past steps. Then, one can choose the next stepsizes ρ_j^k and ϵ_j^k accordingly. Practically, one chooses two nonnegative sequences (θ_ρ^k) and (θ_ϵ^k) which satisfy assumption (6), and one defines iteratively:

$$\forall k\in\mathbb{N},\ \forall j\in\{0,\dots,T\},\ \epsilon_j^k = \theta_\epsilon^{g_j^k},\ \text{and},\ \rho_j^k = \theta_\rho^{g_j^k}.$$

In many cases, (ϵ_j^k) and (ρ_j^k) will satisfy assumption (6), but since ϵ_j^k and ρ_j^k depend on (X_j^{k+1}, U_j^{k+1}), i.e., are, in one sense, anticipative, we fail to prove that this heuristic leads to the convergence of algorithm II.1.

To sum up, our idea is to ensure that the stepsizes decrease according to the frequency each neighborhood has been explored. Therefore, rarely explored regions should have a slower decreasing speed for the corresponding window sizes and depths, and vice versa for frequently visited regions.

III.4. **On the curse of dimensionality.** Beside the computation time limitations, we can also have a look at the dimensional limitations of our approach. Intuitively, our algorithm will take longer when the dimension of the state space will increase. Assumption (4) links the bandwidth of the kernel with the step size ϵ^k. In the case of Gaussian kernels (see eq. (3)) of bandwidth η^k, assumption (4) typically implies that $\epsilon^k \approx \left(\eta^k\right)^d$, where d is the dimension of the state space. Therefore, in a high dimensional setting, the bandwidth will be required to decrease very slowly to ensure enough exploration of the state space by the drawings, before finer details of the Q functions are estimated using kernels of small bandwidths.

IV. Numerical applications

In this section we consider two classical numerical applications of our algorithm, one, the Mountain Car Task [16, example 8.2], being deterministic, and the other, the Puddle World Task [14], being slightly stochastic. Numerical applications of this algorithm on highly stochastic option pricing problems are discussed by the authors in [1].

Numerical applications follow the framework described in [6] so that results can be directly compared with, e.g., those published in [9].

IV.1. **The Mountain Car Task.** Consider the task of driving an underpowered car up a steep mountain road [16, example 8.2]. The difficulty is that gravity is stronger than the car's engine, and even at full throttle the car cannot accelerate up the steep slope. The car has to move away from the goal until it has enough inertia to carry it up forward the steep slope even though it is slowing down the whole way. This is a simple example of a continuous control task where things have to get worse in a sense (farther from the goal) before they can get better. Many control methodologies have great difficulties with tasks of this kind unless explicitly aided by a human designer.

There are three possible actions: full throttle forward (+1), full throttle reverse (−1), and zero throttle (0). The car moves according to a simplified physics. Its position, x_t, and velocity, $\dot{x}_t$, are updated by

$$\begin{aligned} (7) \qquad x_{t+1} &= \text{bound}\left[x_t + \dot{x}_{t+1}\right], \\ (8) \qquad \dot{x}_{t+1} &= \text{bound}\left[\dot{x}_{t+1} + 0.001a_t - 0.0025\cos\left(3x_t\right)\right], \end{aligned}$$

where the bound enforces $-1.2 \leq x_{t+1} \leq 0.5$ and $-0.07 \leq \dot{x}_{t+1} \leq 0.07$. When x_{t+1} reaches the left bound, $\dot{x}_{t+1}$ is reset to zero. When it reaches the right bound, the goal is reached and the episode is terminated. Each episode starts from a random position and velocity uniformly chosen from its feasibility ranges.

To clarify, let us introduce the state variable $s_t = (x_t, \dot{x}_t)$. The problem can thus be written as a minimization problem:

$$\left\{ \begin{array}{ll} \min\limits_{T\in\mathbb{N},(a_t)_{t\leq T}\in\{-1,0,1\}^{T+1}} & T \\ & \\ s_{t+1} & = f\left(s_t, a_t\right), \\ s_0 & = s, \\ s_T & = S^*, \end{array} \right.$$

where T denotes the arrival time, S^* denotes the goal area and f denotes the transportation equations (7). Then we introduce the mapping Q defined by:

$$Q\left(s,a\right) = \left\{ \begin{array}{ll} 1 + \min\limits_{a'} Q\left(f\left(s,a\right), a'\right) & \text{if } s \notin S^*, \\ 0 & \text{if } s \in S^*. \end{array} \right.$$

Then the update in the algorithm can be summed up as follows:

$$Q^{k+1}\left(\cdot,\cdot\right) = Q^k\left(\cdot,\cdot\right) + \rho_j^k \Delta^{k+1} K_j^k\left(s^{k+1}, a^{k+1}, \cdot, \cdot\right),$$

with:

$$\Delta^{k+1} = \left\{ \begin{array}{l} \left[1 + \min\limits_{a'} Q^k\left(f\left(s^{k+1}, a^{k+1}\right), a'\right)\right] - Q^k\left(s^{k+1}, a^{k+1}\right) \\ \text{if } s^{k+1} \notin S^*, \\ 0 - Q^k\left(s^{k+1}, a^{k+1}\right) \qquad \text{otherwise.} \end{array} \right.$$

The algorithm randomly tries all possible strategies and updates the expected time left to the goal by being at state s^k and applying control a^k.

Our implementation makes use of the on-policy drawings and stepsizes described in subsections III.2 and III.3. Greediness is fixed at $\alpha = 0.1$. The sequences (ϵ_j^k) and (ρ_j^k) (or in the case of the on-policy stepsizes (θ_ρ^k) and (θ_ϵ^k)), are chosen to be of the form:

$$\rho_j^k = \frac{1}{a_\rho + b_\rho k^{-c_\rho}},\ \epsilon_j^k = \frac{1}{a_\epsilon + b_\epsilon k^{-c_\epsilon}},$$

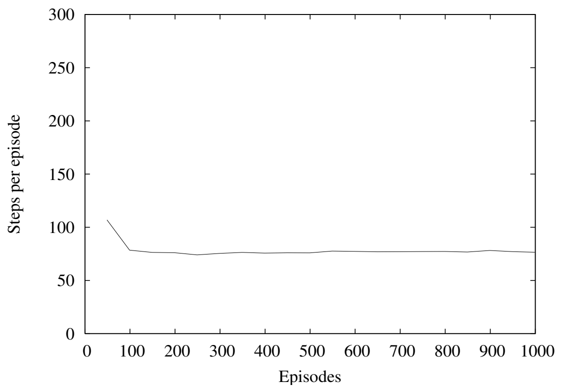
with a_ρ, b_ρ, c_ρ, a_ϵ, b_ϵ and c_ϵ appropriately chosen constants to satisfy assumption (6). The constants are chosen empirically using independent trials. Each episode is limited to at most 300 steps.

We reproduce the benchmarks used in Jong and Stone [9]. The first curve on the left on figure 1 represents the number of steps required to reach the goal, averaged over 50 starting points (random position, and zero velocity) defined at the beginning of the algorithm and used cyclically for each episode. The second curve on the left represents the number of steps required to reach the goal, averaged of 50 independent trials, for which the episodes start using a random position and a zero velocity. We observe that after about 50 episodes, the strategy is nearly optimal.

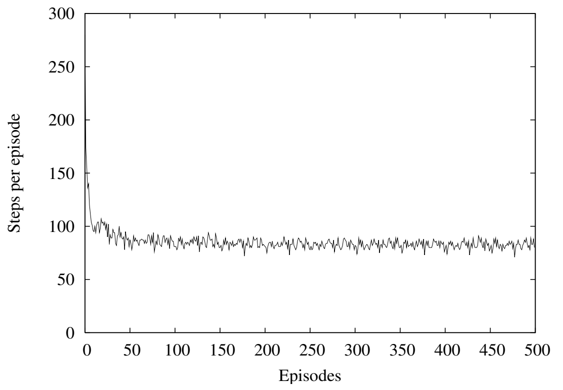
The performance of our algorithm and the one presented by Jong and Stone in [9] seem similar.

IV.2. **The Puddle World Task.** In the Puddle World Task described in [14], an agent starting randomly in a square has to reach one of the corners while avoiding two oval puddles. The state of the agent is represented by its coordinates $(x, y) \in [0,1]^2$. There are four possible actions: up, down, right, and left, which moves the agent 0.05 in the corresponding direction, with independent Gaussian noises of standard deviation 0.01 added to both x and y. The reward is -1 for each step, plus a penalty if either or both of the two puddles are entered, equal to -400 times the distance into the puddle, i.e., the distance to the nearest edge. The puddles extend with radius 0.1 from two line segments: $(0.1, 0.75)$ to $(0.45, 0.75)$ and $(0.45, 0.4)$ to $(0.45, 0.8)$. The goal region is defined by $x + y \geq 2 \times 0.95$.

Similarly to the previous example, the first curve on the right on figure 2 represents the number of steps required to reach the goal, averaged over 50 starting points (random position) defined at the beginning of the algorithm and used cyclically for each episode. The second curve on the right represents the number of steps required to reach the goal, averaged of 50 independent trials, for which the episodes start using a random position. We observe that after about 25 episodes, the strategy is nearly optimal. As for the mountain car task, the performance of our algorithm and the one presented by Jong and Stone in [9] seem similar.

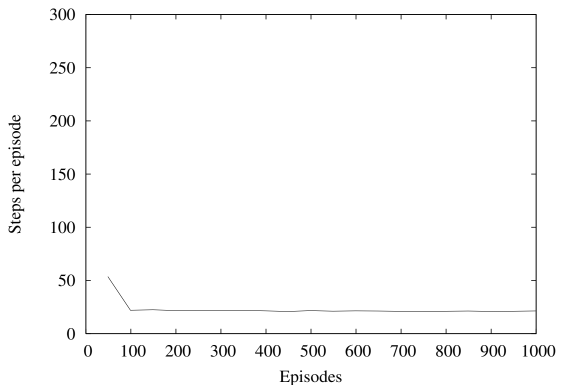


Average over 50 episodes

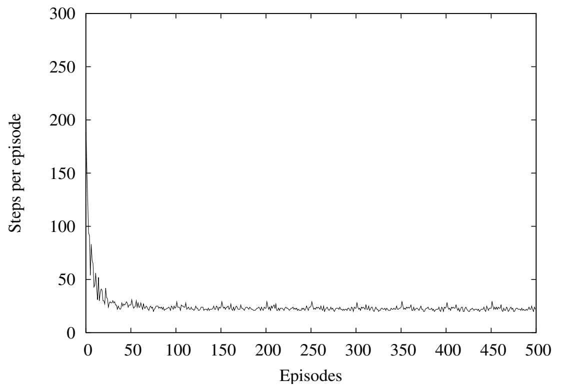


Average over 50 independent trials

FIGURE 1. Learning curves for Mountain Car



Average over 50 episodes



Average over 50 independent trials

FIGURE 2. Learning curves for Puddle World

REFERENCES

[1] K. Barty, P. Girardeau, J.-S. Roy, and C. Strugarek. Application of kernel-based stochastic gradient algorithms to option pricing. *submitted to Monte Carlo Methods and Applications*, 2005.

[2] K. Barty, P. Girardeau, J.-S. Roy, and C. Strugarek. A Q-learning algorithm with continuous state space. submitted to Journal of Machine Learning Reasearch, 2006.

[3] K. Barty, J.-S. Roy, and C. Strugarek. A stochastic gradient type algorithm for closed loop problems. *submitted to Mathematical Programming*, 2005.

[4] K. Barty, J.-S. Roy, and C. Strugarek. Hilbert-valued perturbed subgradient algorithms. 2006. accepted for publication in Math of Operation Research.

[5] D.P. Bertsekas and J.N. Tsitsiklis. Gradient convergence in gradient methods. *SIAM J. Optim.*, 10(3):627–642, 2000.

[6] A. Dutech, T. Edmunds, J. Kok, M. G. Lagoudakis, M. Littman, M. Riedmiller, B. Russell, B. Scherrer, R. Sutton, S. Timmer, N. Vlassis, A. White, and S. Whiteson, editors. *Reinforcement learning benchmarks and bake-offs II*, 2005.

[7] L. Greengard and J. Strain. The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.

[8] T. Jaakkola, M.I. Jordan, and S.P. Singh. Convergence of stochastic iterative dynamic programming algorithms. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 703–710. Morgan Kaufmann Publishers, Inc., 1994.

[9] N.K. Jong and P. Stone. Kernel-based models for reinforcement learning. The ICML-2006 Workshop on Kernel Methods in Reinforcement Learning, June 2006.

[10] T.L. Lai. Stochastic Approximation. *Ann. Stat.*, 31(2):391–406, 2003.

[11] D. Ormoneit and P. Glynn. Kernel-based reinforcement learning in average-cost problems: An application to optimal portfolio choice. In *NIPS*, pages 1068–1074, 2000.

[12] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. Technical Report TR 1999-8, Statistics, Stanford University, 1999.

[13] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[14] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. The MIT Press, 1996.

[15] R.S. Sutton. Learning to predict by the method of temporal difference. *IEEE Trans. Autom. Control*, 37:332–341, 1988.

[16] R.S. Sutton and A.G. Barto. *Reinforcement Learning, an Introduction.* MIT press Cambridge, 1998.

[17] J.N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16:185–202, 1993.

[18] C. Watkins. *Learning from delayed rewards.* PhD thesis, King's College, Cambridge, 1989.

[19] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[20] C. Yang, R. Duraiswami, N. Gumerov, and L. Davis. Improved fast gauss transform and efficient kernel density estimation. *IEEE International Conference on Computer Vision*, pages 464–471, 2003.