Iteratively Extending Time Horizon Reinforcement Learning

Damien Ernst[⋆], Pierre Geurts[⋆⋆], and Louis Wehenkel

Department of Electrical Engineering and Computer Science
Institut Montefiore, University of Liège
Sart-Tilman B28, B4000 Liège, Belgium
{ernst,geurts,lwh}@montefiore.ulg.ac.be

Abstract. Reinforcement learning aims to determine an (infinite time horizon) optimal control policy from interaction with a system. It can be solved by approximating the so-called Q-function from a sample of four-tuples (x_t, u_t, r_t, x_{t+1}) where x_t denotes the system state at time t, u_t the control action taken, r_t the instantaneous reward obtained and x_{t+1} the successor state of the system, and by determining the optimal control from the Q-function. Classical reinforcement learning algorithms use an ad hoc version of stochastic approximation which iterates over the Q-function approximations on a four-tuple by four-tuple basis. In this paper, we reformulate this problem as a sequence of *batch mode* supervised learning problems which in the limit converges to (an approximation of) the Q-function. Each step of this algorithm uses the full sample of four-tuples gathered from interaction with the system and extends by one step the horizon of the optimality criterion. An advantage of this approach is to allow the use of standard batch mode supervised learning algorithms, instead of the incremental versions used up to now. In addition to a theoretical justification the paper provides empirical tests in the context of the "Car on the Hill" control problem based on the use of ensembles of regression trees. The resulting algorithm is in principle able to handle efficiently large scale reinforcement learning problems.

1 Introduction

Many interesting problems in many fields can be formulated as closed-loop control problems, i.e. problems whose solution is provided by a mapping (or a control policy) $u_t = \mu(x_t)$ where x_t denotes the state at time t of a system and u_t an action taken by a controlling agent so as to influence the instantaneous and future behavior of the system. In many cases these problems can be formulated as *infinite horizon discounted reward discrete-time optimal control problems*, i.e. problems where the objective is to find a (stationary) control policy $\mu^*(\cdot)$ which maximizes the expected return over an infinite time horizon defined as follows:

[⋆] Research Fellow FNRS
[⋆⋆] Postdoctoral Researcher FNRS

$$J_{\infty}^{\mu} = \lim_{N \to \infty} E \left\{ \sum_{t=0}^{N-1} \gamma^t r_t \right\}, \tag{1}$$

where $\gamma \in [0,1[$ is the discount factor, r_t is an instantaneous reward signal which depends only on the state x_t and action u_t at time t, and where the expectation is taken over all possible system trajectories induced by the control policy $\mu(\cdot)$.

Optimal control theory, and in particular dynamic programming, aims to solve this problem "exactly" when the explicit knowledge of system dynamics and reward function are given a priori. In this paper we focus on *reinforcement learning (RL)*, i.e. the use of automatic learning algorithms in order to solve the optimal control problem "approximately" when the sole information available is the one we obtain from system transitions from t to $t+1$. Each system transition provides the knowledge of a new four-tuple (x_t, u_t, r_t, x_{t+1}) of information and we aim here to compute $\mu^*(.)$ from a sample $\mathcal{F} = (x_t^k, u_t^k, r_t^k, x_{t+1}^k), k = 1, \ldots, \ell$ of such four-tuples.

It is important to contrast the RL protocol with the standard batch mode supervised learning protocol, which aims at determining, from the sole information of a sample S of input-output pairs (i,o), a function $h^* \in \mathcal{H}$ ($\mathcal{H}$ is called the hypothesis space of the learning algorithm) which minimizes the expected approximation error, e.g. defined in the case of least squares regression by the following functional:

$$\mathrm{Err}^h = \sum_{(i,o) \in S} |h(i) - o|^2 \,. \tag{2}$$

Notice that the use of supervised learning in the context of optimal control problems would be straightforward if, instead of the sample $\mathcal{F}$ of four-tuples, we could provide the learning algorithm with a sample of input-output pairs $(x, \mu^*(x))$ (see for example [9] for a discussion on the combination of such a scheme with reinforcement learning). Unfortunately, in many interesting control problems this type of information can not be acquired directly, and the specific difficulty in reinforcement learning is to infer a good approximation of the optimal control policy only from the information given in the sample $\mathcal{F}$ of four-tuples. Existing reinforcement learning algorithms can be classified into two categories:

- *Model based RL methods:* they use (batch mode or incremental mode) supervised learning to determine from the sample $\mathcal{F}$ of four-tuples on the one hand an approximation of the system dynamics:

$$f_1(x,u,x') \approx P(x_{t+1} = x' | x_t = x, u_t = u) \tag{3}$$

  and on the other hand an approximation of the expected reward function:

$$f_2(x,u) \approx E\{r_t | x_t = x, u_t = u\}. \tag{4}$$

  Once these two functions have been obtained, model based algorithms derive the optimal control policy by dynamic programming [5, 8].

– *Non-model based RL methods:* they use incremental mode supervised learning in order to determine an approximation of the Q-function associated to the control problem. This function is (implicitly) defined by the following equation (known as the Bellman equation):

$$Q(x,u) = E\left\{ r_t + \gamma \max_{u'} Q(x_{t+1},u') \middle| x_t = x, u_t = u \right\}. \tag{5}$$

The optimal control policy can be directly determined from this (unique) Q-function by the following relation

$$\mu^*(x) = \arg\max_{u} Q(x,u). \tag{6}$$

The most well-known algorithm falling into the latter category is the so-called Q-learning method [11].

Our proposal is based on the observation that neither of these two approaches are able to fully exploit the power of modern supervised learning methods. Indeed, model based approaches are essentially linked to so-called state space discretization which aims at building a finite Markov Decision Problem (MDP) and are strongly limited by the curse of dimensionality: in order to use the dynamic programming algorithms, the state and control spaces need to be discretized and the number of cells of any discretization scheme increases exponentially with the number of dimensions of the state space. Non-model based approaches have, to our best knowledge, been combined only with *incremental* (on-line) learning algorithms (see e.g. [10]).

With respect to these approaches, we propose a novel non-model based RL framework which is able to exploit any *generic batch mode* supervised learning algorithm to model the Q-function. The resulting algorithm is illustrated on a simple problem where it is combined with three supervised learning algorithms based on regression trees. The rest of the paper is organized as follows: Section 2 introduces the underlying idea of our approach and gives a precise description of the proposed algorithm; Section 3 provides a validation in the context of the "Car on the Hill" control problem; Section 4 provides discussions, directions for future research and conclusions.

2 Iteratively Extending Time Horizon in Optimal Control

The approach that we present is based on the fact that the optimal (stationary) control policy of an *infinite* horizon problem can be formalized as the limit of a sequence of *finite* horizon control problems, which can be solved in an iterative fashion by using any standard supervised learning algorithm.

2.1 Iteratively Extending time Horizon in Dynamic Programming

We consider a discrete-time stationary stochastic system defined by its dynamics, i.e. a transition function defined over the Cartesian product $X \times U \times W$ of the state space X, the control space U, and the disturbance space W:

$$x_{t+1} = f(x_t, u_t, w_t), \tag{7}$$

a reward signal also defined over $X \times U \times W$:

$$r_t = r(x_t, u_t, w_t), \tag{8}$$

a noise process defined by a conditional probability distribution:

$$w_t \sim P_w(w = w_t | x = x_t, u = u_t), \tag{9}$$

and a probability distribution over the initial conditions:

$$x_0 \sim P_x(x = x_0). \tag{10}$$

For a given (finite) horizon N, let us denote by

$$\pi_N(t, x) \in U, t \in \{0, \dots, N-1\}; x \in X \tag{11}$$

a (possibly time varying) N-step control policy (i.e. $u_t = \pi_N(t, x_t)$), and by

$$J_N^{\pi_N} = E\{\sum_{t=0}^{N-1} \gamma^t r_t\} \tag{12}$$

the N-step reward of the closed-loop system using this policy. An N-step optimal policy is a policy which among all possible such policies maximizes $J_N^{\pi_N}$ for any P_x on the initial conditions. Notice that (under mild conditions) such a policy always does indeed exist although it is not necessarily unique.

Our algorithm exploits the following properties of N-step optimal policies (these are classical results of dynamic programming theory [1]):

1. The sequence of policies obtained by considering the sequence of Q_i-functions iteratively defined by

$$Q_1(x, u) = E\{r_t | x_t = x, u_t = u\} \tag{13}$$

and

$$Q_N(x, u) = E\left\{ r_t + \gamma \max_{u'} Q_{N-1}(x_{t+1}, u') \middle| x_t = x, u_t = u \right\}, \forall N > 1, \tag{14}$$

and the following two conditions[1]

$$\pi_N^*(0, x) = \arg\max_u Q_N(x, u), \forall N \geq 0 \tag{15}$$

and

$$\pi_N^*(t+1, x) = \pi_{N-1}^*(t, x), \forall N > 1, t \in \{0, \dots, N-2\} \tag{16}$$

is optimal.

2. The sequence of stationary policies defined by $\mu_N^*(x) = \pi_N^*(0, x)$ converges (globally, and for any P_x on the initial conditions) to $\mu^*(x)$ in the sense that

$$\lim_{N \to \infty} J_\infty^{\mu_N^*} = J_\infty^{\mu^*}. \tag{17}$$

3. The sequence of functions Q_N converges to the (unique) solution of the Bellman equation (eqn. (5)).

[1] Actually this definition does not necessarily yield a unique policy, but any policy which satisfies this condition is appropriate, and it is straightforward to define a procedure constructing such a policy from the sequence of Q_i-functions.

2.2 Iteratively Extending Time Horizon in Reinforcement Learning

The proposed algorithm is based on the use of supervised learning in order to produce a sequence $\hat{Q}_i$ of approximations of the Q_i-functions defined above, by exploiting at each step the full sample of four-tuples $\mathcal{F} = (x_t^k, u_t^k, r_t^k, x_{t+1}^k), k = 1, \dots, \ell$ in batch mode together with the function produced at the preceding step.

Initialization. The algorithm starts by using the sample $\mathcal{F}$ of four-tuples in order to construct an approximation of $Q_1(x, u)$. This can be achieved using the x_t, u_t components of each four-tuple as inputs, and the r_t component as output and by using a supervised regression algorithm in order to find in its hypothesis space $\mathcal{H}$ a function satisfying

$$\hat{Q}_1 = \arg\min_{h \in \mathcal{H}} \sum_{k=1}^{\ell} |h(x_t^k, u_t^k) - r_t^k|^2. \tag{18}$$

Iteration. Step i ($i > 1$) of the algorithm uses the function produced at step $i-1$ to modify the output of each input-output pair associated to each four-tuple by

$$o_i^k = r_t^k + \gamma \max_{u'} \hat{Q}_{i-1}(x_{t+1}^k, u') \tag{19}$$

and then applies the supervised learning algorithm to build

$$\hat{Q}_i = \arg\min_{h \in \mathcal{H}} \sum_{k=1}^{\ell} |h(x_t^k, u_t^k) - o_i^k|^2. \tag{20}$$

Stopping Conditions. For the theoretical sequence of policies an error bound on the sub-optimality in terms of the number of iterations is given by the following equation

$$|J_\infty^{\mu_N^*} - J_\infty^{\mu^*}| < \frac{\gamma^N B_r}{1 - \gamma}, \tag{21}$$

where $B_r > \sup r(x, u, w)$. This equation can be used to fix an upper bound on the number of iterations for a given a priori fixed optimality gap.

Another possibility is to exploit the convergence property of the sequence of Q_i-functions in order to decide when to stop the iteration, e.g. when

$$|\hat{Q}_N - \hat{Q}_{N-1}| < \epsilon. \tag{22}$$

Control Policy Derivation. The final control policy seen as an approximation of the optimal stationary closed-loop policy is in principle derived by

$$\hat{\mu}^*(x) = \hat{\mu}_N^*(x) = \arg\max_u \hat{Q}_N(x, u). \tag{23}$$

If the control space is finite, this can be done using exhaustive search. Otherwise, the algorithm to achieve this will depend on the type of approximation architecture used.

Consistency. It is interesting to question under which conditions this algorithm provides consistency, i.e. under which conditions the sequence of policies generated by our algorithm and using a sample of increasing size would converge to the optimal control policy within a pre-specified optimality gap. Without any assumption on the used supervised learning algorithm and on the sampling mechanism nothing can be said about consistency. On the other hand, if each one of the true Q_i-functions can be arbitrarily well approximated by a function of the hypothesis space and if the sample (in asymptotic regime) contains an infinite number of times each possible state-action pair (x, u), then consistency is ensured trivially. Further research is necessary in order to determine less ideal assumptions both on the hypothesis space and on the sampling mechanism which would still guarantee consistency.

Solution Characterization. Another way to state the reinforcement learning problem would consist of defining the approximate Q-function as the solution of the following equation

$$\hat{Q} = \arg\min_{h\in\mathcal{H}} \sum_{k=1}^{l} \left| h(x_t^k, u_t^k) - \left(r_t^k + \gamma \max_{u'} h(x_{t+1}^k, u') \right) \right|^2 . \quad (24)$$

Our algorithm can be viewed as an iterative algorithm to solve this minimization problem starting with an initial guess $Q_0(x, u) \equiv 0$ and at each iteration $i > 0$ updating the function according to

$$\hat{Q}_i = \arg\min_{h\in\mathcal{H}} \sum_{k=1}^{l} \left| h(x_t^k, u_t^k) - \left(r_t^k + \gamma \max_{u'} \hat{Q}_{i-1}(x_{t+1}^k, u') \right) \right|^2 . \quad (25)$$

2.3 Supervised Regression Algorithm

In principle, the proposed framework can be combined with any available supervised learning method designed for regression problems. In order to be practical, the desirable features of the used algorithm are as follows:

– *Computational efficiency and scalability of the learning algorithm.* Specially with respect to sample size and dimensionality of the state space X and the control space U.
– *Modeling flexibility.* The Q_i-functions to be modeled by the algorithm are unpredictable in shape; hence no prior assumption can be made on the parametric shape of the approximation architecture, and the automatic learning algorithm should be able to adapt its model by itself to the problem data.
– *Reduced variance,* in order to work efficiently in small sample regimes.
– *Fully automatic operation.* The algorithm may be called several hundred times and it is therefore not possible to ask for a human operator to tune some meta-parameters at each step of the iterative procedure.
– *Efficient use of the model,* in order to derive the control from the Q-function.

In the simulation results given in the next section, we have compared three learning algorithms based on regression trees which we think offer a good compromise in terms of the criteria established above. We give a very brief description of each variant below.

Regression Trees. Classification and regression trees are among the most popular supervised learning algorithms. They combine several characteristics such as interpretability of the models, efficiency, flexibility, and fully automatic operation which make them particularly attractive for this application. To build such trees, we have implemented the CART algorithm as described in [4].

Tree Bagging. One drawback of regression trees is that they suffer from a high variance. Bagging [2] is an ensemble method proposed by Breiman that often improves very dramatically the accuracy of trees by reducing their variance. With bagging, several regression trees are built, each from a different bootstrap sample drawn from the original learning sample. To make a prediction with this set of M trees, we simply take the average predictions of these M trees. Note that, while bagging inherits several advantages of regression trees, it increases their computing times significantly.

Extremely Randomized Trees (Extra-trees). Besides bagging, several other methods to build tree ensembles have been proposed that often improve the accuracy with respect to tree bagging (e.g. random forests [3]). In this paper, we propose to evaluate our own recent proposal which is called "Extra-trees". Like bagging, this algorithm works by taking the average predictions of several trees. Each of these trees is built from the the original learning sample by selecting its tests fully at random. The main advantages of this algorithm with respect to bagging is that it is computationally much faster (because of the extreme randomization) and also often more accurate. For more details about this algorithm, we refer the interested reader to [6, 7].

3 Illustration: "Car on the Hill" Control Problem

The precise definition of the test problem is given in the appendix. It is a version of a quite classical test problem used in the reinforcement learning literature.

A car is traveling on a hill (the shape of which is given by the function $H(p)$ of Figure 3b). The objective is to bring the car in minimal time to the top of the hill ($p = 1$ in Figure 3b). The problem is studied in discrete-time, which means here that the control variable can be changed only every $0.1\,s$. The control variable acts directly on the acceleration of the car (eqn. (27), appendix) but can only assume two extreme values (full acceleration or full deceleration). The reward signal is defined in such a way that the infinite horizon optimal control policy is a minimum time control strategy (eqn. (29), appendix).

Our test protocol uses an "off-line" learning strategy. First, samples of four-tuples are generated from fixed initial conditions and random walk in the control space. Then these samples are used to infer control strategies according to the proposed method. Finally these control strategies are assessed.

3.1 Four-Tuples Generation

To collect the samples of four-tuples, we observed a number of episodes of the system. All episodes start from the same initial state corresponding to the car stopped at the bottom of the valley (i.e. $(p, s) = (-0.5, 0)$) and stop when the car leaves the region of the state space depicted in Figure 3a. In each episode, the action u_t at each time step is chosen at random with equal probability among its two possible values $u = -4$ and $u = 4$. We will consider hereafter three different samples of four-tuples denoted by $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ containing respectively the four-tuples obtained after 1000, 300, and 100 episodes. These samples are such that $\#\mathcal{F}_1 = 58089$, $\#\mathcal{F}_2 = 18010$, and $\#\mathcal{F}_3 = 5930$. Note also that after 100 episodes the reward $r(x_t, u_t, w_t) = 1$ (corresponding to the goal state at the top of the hill) has been observed only 1 time, 5 times after 300 episodes, and 18 times after 1000 episodes.

3.2 Experiments

To illustrate the behavior of the algorithm, we first use the sample $\mathcal{F}_1$ with Extra-trees[2] As the action space is binary, we choose to separately model the functions $\hat{Q}_N(x, -4)$ and $\hat{Q}_N(x, 4)$ by two ensembles of 50 Extra-trees. The policy $\hat{\mu}_1^*$ obtained is represented in Figure 1a. Black bullets represent states for which $\hat{Q}_1(x, -4) > \hat{Q}_1(x, 4)$, white bullets states for which $\hat{Q}_1(x, -4) < \hat{Q}_1(x, 4)$, and grey bullets states for which $\hat{Q}_1(x, -4) = \hat{Q}_1(x, 4)$. Successive policies $\hat{\mu}_N^*$ for increasing N are given on Figures 1b-1f. After 50 iterations $\hat{\mu}_N^*$ has almost stabilized.

To associate a score to each policy $\hat{\mu}_N^*$, we define a set $X' : X' = \{(p, s) \in X | \exists i, j \in \mathbb{Z} | (s, p) = (0.125 * i, 0.375j)\}$ and estimate the value of $J_\infty^{\hat{\mu}_N^*}$ when $P_x(x_0) = \frac{1}{\#X'}$ if $x_0 \in X'$ and 0 otherwise. The evolution of the score for increasing N is represented in Figure 2a for the three learning algorithms. With Bagging and Extra-trees, we average 50 trees. After about 20 episodes, the score does not improve anymore. Comparing the three supervised learning algorithms, it is clear that bagging and Extra-trees are superior to single regression trees. Bagging and Extra-trees are very close to each other but the score of Extra-trees grows faster and is also slightly more stable.

On Figure 2b, we compare score curves corresponding to the three different sample sizes (with Extra-trees). As expected, we observe that a decrease of the number of four-tuples decreases the score.

To give a better idea of the quality of the control strategy induced by our algorithm, it would be interesting to compare it with the optimal one. Although it is difficult to determine analytically the optimal control policy for this problem, it is however possible to determine $J_\infty^{\mu^*}$ when the probability distribution on the initial states is such that $P_x(x_0 = x) = 1$ if x corresponds to the state

[2] The results with regression trees and tree bagging are discussed afterwards.

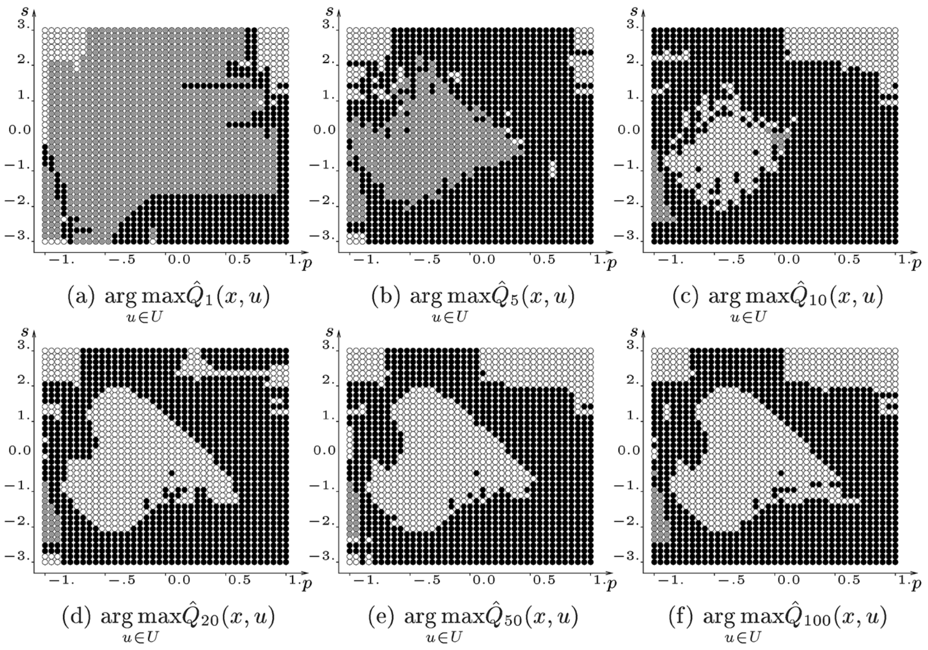


Fig. 1. Representation of $\hat{\mu}_N^*$ for different values of N. Sample $\mathcal{F}_1$

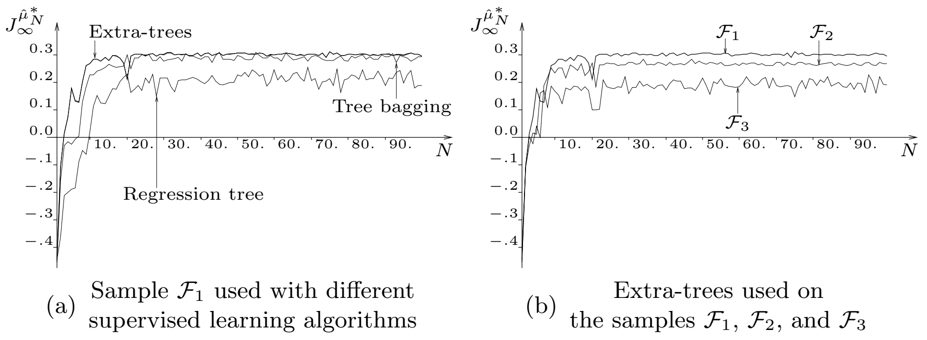


Fig. 2. Evaluation of the policy $\hat{\mu}_N^*$

$(p,s) = (-0.5, 0)$ and 0 otherwise. This is achieved by exhaustive search, trying out all possible control sequences of length k when the system initial state is $(p,s) = (-0.5, 0)$ and determining the smallest value of k for which there is a control sequence that leads the car on the top of the hill. From this procedure, we find a minimum value of $k = 19$ and then $J_\infty^{\mu^*} = 0.397214$ $(= \gamma^{k-1})$. If we use from the same initial state the policy learned by our algorithm (with Extra-trees), we get:

– from $\mathcal{F}_1$, $J_\infty^{\hat{\mu}_{100}^*} = 0.397214 = \gamma^{18}$
– from $\mathcal{F}_2$, $J_\infty^{\hat{\mu}_{100}^*} = 0.397214 = \gamma^{18}$
– from $\mathcal{F}_3$, $J_\infty^{\hat{\mu}_{100}^*} = 0.358486 = \gamma^{20}$

For the two largest samples, $J_\infty^{\hat{\mu}_{100}^*}$ is equal to the optimum value $J_\infty^{\mu^*}$ while it is only slightly inferior in the case of the smallest one.

3.3 Comparison with a Non-model Based Incremental Algorithm

It is interesting to question whether our proposed algorithm is more efficient in terms of learning speed than non-model based iterative reinforcement learning algorithms. In an attempt to give an answer to this question we have considered the standard Q-learning algorithm with a regular grid as approximation architecture[3].

We have used this algorithm during 1000 episodes (the same as the ones used to generate $\mathcal{F}_1$) and then we have extracted from the resulting approximate Q-function the policy $\hat{\mu}^*$ and computed $J_\infty^{\hat{\mu}^*}$ when considering the same probability distribution on the initial states as the one used to compute the values of $J_\infty^{\hat{\mu}_N^*}$ represented on Figures 2a-b. The highest value of $J_\infty^{\hat{\mu}^*}$ so obtained by repeating the process for different grid sizes (a 10×10, a 11×11, $\cdots$ and a 100×100 grid) is 0.039 (which occurs for a 13×13 grid). This value is quite small compared to $J_\infty^{\hat{\mu}_{100}^*} = 0.295$ obtained when using $\mathcal{F}_1$ as sample and the Extra-trees as regression method (Figure 2a). Even when using ten times more (i.e. $10,000$) episodes with the Q-learning algorithm, the highest value of $J_\infty^{\hat{\mu}^*}$ obtained over the different grids is still inferior (it is equal to 0.232 and occurs for a 24×24 grid).

4 Discussion and Conclusions

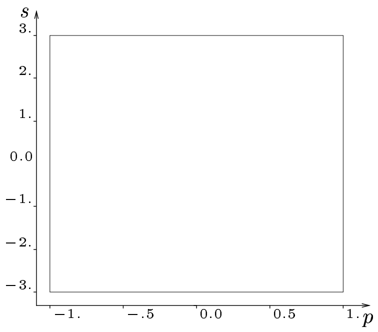
We have presented a novel way of using batch mode supervised learning algorithms efficiently in the context of non-model based reinforcement learning. The resulting algorithm is fully autonomous and has been applied to an illustrative problem where it worked very well.

Probably the most important feature of this algorithm is that it can scale very easily to high dimensional problems (e.g. problems with a large number of input variables and continuous control spaces) by taking advantage of recent advances of supervised learning techniques in this direction. This feature can for example be exploited to handle more easily partially observable problems, where it is necessary to use as inputs a history of observations rather than just the current state. It could also be exploited to carry out reinforcement learning based on perceptual input information (tactile sensors, images, sounds) without requiring complex pre-processing.

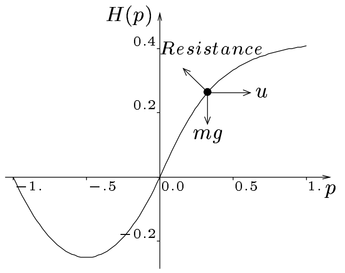
[3] The degree of correction α used in the algorithm has been chosen equal to 0.1 and the Q-function has been initialized to zero everywhere at the beginning of the learning.

Although we believe that our approach to reinforcement learning is very promising, there are still many open questions. In the formulation of our algorithm, we have not made any assumption about the way the four-tuples are generated. However, the quality of the induced control policy depends obviously on the sampling mechanism. So, an interesting future research direction is the determination for a given problem of the smallest possible (for computational efficiency reasons) sample of four-tuples that gives a near optimal control policy. This will raise the related question of how to interact at best with a system so as to generate a good sample of four-tuples. One very interesting property of our algorithm is that these questions are decoupled from the question of the determination of the optimal control policy from a given sample of four-tuples.

Appendix: Precise Definition of the "Car on the Hill" Control Problem



(a) $X \setminus \{x^t\}$



(b) Representation of $H(p)$ (shape of the hill) and of the different forces applied to the car

Fig. 3. The "Car on the Hill" control problem

System dynamics: The system has a continuous-time dynamics described by these two differential equations:

$$\dot{p} = s \tag{26}$$

$$\dot{s} = \frac{u}{m(1+H'(p)^2)} - \frac{gH'(p)}{1+H'(p)^2} - \frac{s^2H'(p)H''(p)}{1+H'(p)^2} \tag{27}$$

where m and g are parameters equal respectively to 1 and 9.81 and where $H(p)$ is a function of p defined by the following expression:

$$H(p) = \begin{cases} p^2 + p & \text{if} \quad p < 0 \\ \frac{p}{\sqrt{1+5p^2}} & \text{if} \quad p \geq 0 \end{cases} \tag{28}$$

The discrete-time dynamics is obtained by discretizing the time with the time between t and $t+1$ chosen equal to $0.100\,s$.

If p_{t+1} and s_{t+1} are such that $|p_{t+1}| > 1$ or $|s_{t+1}| > 3$ then a terminal state x^t is reached.
State space: The state space X is composed of $\{(p, s) \in \mathbb{R}^2 | \, |s| \leq 1 \, \text{and} \, |p| \leq 3\}$ and of a terminal state x^t. $X \setminus \{x^t\}$ is represented on Figure 3a.
Action space: The action space $U = \{-4, 4\}$
Reward function: The reward function $r(x, u, w)$ is defined through the following expression:

$$r(x_t, w_t, u_t) = \begin{cases} -1 & \text{if} \quad p_{t+1} < -1 \quad \text{or} \quad |s_{t+1}| > 3 \\ 1 & \text{if} \quad p_{t+1} > 1 \quad \text{and} \quad |s_{t+1}| \leq 3 \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

Decay factor: The decay factor γ has been chosen equal to 0.95. Notice that in this particular problem the value of γ actually does not influence the optimal control policy.

References

1. D. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, 2nd edition, 2000.
2. L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
3. L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.
4. L. Breiman, J. Friedman, R. Olsen, and C. Stone. *Classification and Regression Trees*. Wadsworth International (California), 1984.
5. D. Ernst. *Near optimal closed-loop control. Application to electric power systems.* PhD thesis, University of Liège, Belgium, March 2003.
6. P. Geurts. *Contributions to decision tree induction: bias/variance tradeoff and time series classification.* PhD thesis, University of Liège, Belgium, May 2002.
7. P. Geurts. Extremely randomized trees. Technical report, University of Liège, 2003.
8. A. Moore and C. Atkeson. Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning*, 13:103–130, 1993.
9. M. T. Rosenstein and A. G. Barto. Supervised learning combined with an actor-critic architecture. Technical report, University of Massachusetts, Department of Computer Science, 2002.
10. W. Smart and L. Kaelbling. Practical Reinforcement Learning in Continuous Spaces. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 2000.
11. C. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.