Selecting concise sets of samples for a reinforcement learning agent

D. Ernst
Dept of Electrical Engineering and Computer Science - University of Liège -Belgium
Email: ernst@montefiore.ulg.ac.be

Abstract

We derive an algorithm for selecting from the set of samples gathered by a reinforcement learning agent interacting with a deterministic environment, a concise set from which the agent can extract a good policy.

The reinforcement learning agent is assumed to extract policies from sets of samples by solving a sequence of standard supervised learning regression problems. To identify concise sets, we adopt a criterion based on an error function defined from the sequence of models produced by the supervised learning algorithm.

We evaluate our approach on two-dimensional maze problems and show its good performances when problems are continuous.

1 Introduction

Generalization of the information in reinforcement learning (RL) has been an open problem for years. Recently, several authors have advocated addressing this problem by solving a sequence of standard supervised learning problems [1, 6]. They have shown that by using non-parametric supervised learning methods and in particular ensembles of regression trees, this framework could lead to excellent generalization performances while avoiding convergence problems of the sequence.

The training sets for the different supervised learning problems contain a number of elements equal to the number of samples the reinforcement learning agent has acquired from interaction with the system. After a certain time of interaction, these samples may become so numerous that this framework may become computationally impractical. To reduce the computational burdens, we propose to select a concise set of sufficiently rich representatives of the samples.

We propose in this paper an algorithm that identifies such concise sets. The algorithm works iteratively by associating to the solution computed from the already selected samples an error function and by selecting the sample for which this error function takes its largest value.

Our algorithm is inspired by work in the classical supervised learning framework. The closest one is probably the certainty-based algorithm from [4] which is applied to classification problems and grows a concise training set by selecting elements with the lowest annotation certainty, while in [7] the selection is rather based on the decrement of the error. Our approach is also related to some techniques in dynamic programming (DP) that iteratively discretize the state space (see e.g. [3, 5]). Our problem is however different since it does not focus on where to generate some new information but well on which information among a given set should be selected. Another fundamental difference is that these DP techniques mainly associate to every discretization a finite Markov Decision Problem from which they deduce an approximate solution to the optimal control problem, whereas here we compute from a given set of samples an approximate solution by solving a sequence of supervised learning problems.

Section 2 formalizes the problem of learning from a set of samples, reviews some classical results of the dynamic programming theory and presents the fitted Q iteration algorithm that solves this problem by reformulating it as a sequence of standard supervised learning problems. The material of this section is largely borrowed from [1] and is valid both for stochastic and deterministic environments. In Section 3 we present our approach to select concise sets of samples when a deterministic environment is assumed. Section 4 gathers simulation results and, finally, Section 5 concludes.

2 Learning from a set of samples

2.1 Problem formulation

Let us consider a system having a *discrete-time dynamics* described by:

$$x_{t+1} = f(x_t, u_t, w_t) \quad t = 0, 1, \cdots \tag{1}$$

where for all t, the state x_t is an element of the state space X, the action u_t is an element of the action space U and the random disturbance w_t an element of the

disturbance space W. The disturbance w_t is generated by the time-invariant conditional probability distribution $P_w(w|x,u)$.

To the transition from t to $t+1$ is associated an instantaneous *reward signal* $r_t = r(x_t, u_t, w_t)$ where $r(x,u,w)$ is the reward function bounded by some constant B_r.

Let $\mu(\cdot) : X \to U$ denote a stationary control policy and J^μ denote the expected return obtained over an infinite time horizon when the system is controlled using this policy (i.e. when $u_t = \mu(x_t), \forall t$). For a given initial condition $x_0 = x$, J^μ is defined as follows:

$$J^\mu(x) = \lim_{N\to\infty} \underset{\substack{w_t \\ t=0,1,\cdots,N-1}}{E} [\sum_{t=0}^{N-1} \gamma^t r(x_t, \mu(x_t), w_t) | x_0 = x] \quad (2)$$

where γ is a discount factor ($0 \leq \gamma < 1$) that weighs short-term rewards more than long-term ones, and where the conditional expectation is taken over all trajectories starting with the initial condition $x_0 = x$. Our objective is to find an optimal stationary policy μ^*, i.e. a stationary policy that maximizes J^μ for all x. The only information that we assume available to solve this problem is the one obtained from the observation of a certain number of one-step system transitions (from t to $t+1$). Each system transition provides the knowledge of a new sample (x_t, u_t, r_t, x_{t+1}) that we name *four-tuple*. Since it is usually not possible to determine an optimal policy from a finite number of four-tuples, we aim at computing an approximation of μ^* from a set $\mathcal{F} = \{(x_t^l, u_t^l, r_t^l, x_{t+1}^l),\ l = 1, \cdots, \#\mathcal{F}\}$ of such four-tuples.

2.2 Dynamic programming results

Let H denote the mapping that transforms any bounded function $K : X \times U \to \mathbb{R}$ into

$$(HK)(x,u) = \underset{w}{E}[r(x,u,w) + \gamma \max_{u' \in U} K(f(x,u,w), u')] \quad (3)$$

where the expectation is computed by using $P(w) = P_w(w|x,u)$.
The sequence of Q_N-functions defined on $X \times U$

$$Q_N(x,u) \quad = \quad (HQ_{N-1})(x,u) \quad \forall N > 0 \quad (4)$$

with $Q_0(x,u) \equiv 0$ converges, in infinity norm, to the Q-function, defined as the (unique) solution of the Bellman equation:

$$Q(x,u) \quad = \quad (HQ)(x,u). \quad (5)$$

A policy μ^* that satisfies

$$\mu^*(x) = \arg\max_{u \in U} Q(x,u) \quad (6)$$

is an optimal stationary policy.

Let us denote by μ_N^* the stationary policy

$$\mu_N^*(x) = \arg\max_{u \in U} Q_N(x,u). \quad (7)$$

The following bound on the suboptimality of μ_N^* holds:

$$\|J^{\mu^*} - J^{\mu_N^*}\|_\infty \leq \frac{2\gamma^N B_r}{(1-\gamma)^2} \quad . \quad (8)$$

2.3 Fitted Q iteration

The fitted Q iteration algorithm computes from the set of four-tuples $\mathcal{F}$ the functions $\hat{Q}_1, \hat{Q}_2, \cdots, \hat{Q}_N$, approximations of the functions $Q_1, Q_2 \cdots, Q_N$ defined by Eqn (5), by solving a sequence of standard supervised learning regression problems. The policy

$$\hat{\mu}_N^*(x) = \arg\max_{u \in U} \hat{Q}_N(x,u) \quad (9)$$

is then taken as approximation of the optimal stationary policy. The training set for the kth problem ($k \geq 1$) of the sequence is

$$((x_t^l, u_t^l), r_t^l + \gamma \max_{u \in U} \hat{Q}_{k-1}(x_{t+1}^l, u)), l = 1, \cdots, \#\mathcal{F} \quad (10)$$

with $\hat{Q}_0(x,u) = 0$ everywhere. The supervised learning regression algorithm produces from this training set the function $\hat{Q}_k$ that is used to determine the next training set and from there, the next function of the sequence.

We have the following upper bound on the suboptimality of $\hat{\mu}_N^*$:

$$\|J^{\mu^*} - J^{\hat{\mu}_N^*}\|_\infty \leq \frac{2\gamma^N B_r + \|\epsilon\|_\infty (2 - 2\gamma^N)}{(1-\gamma)^2} \quad (11)$$

where the error function $\epsilon : X \times U \to \mathbb{R}$ is:

$$\epsilon(x,u) = \max_{k \in \{1,2,\cdots,N\}} |\hat{Q}_k(x,u) - (H\hat{Q}_{k-1})(x,u)|. \quad (12)$$

3 Concise set of samples selection

The algorithm. The algorithm we propose to select from $\mathcal{F}$ the n most informative four-tuples works iteratively. At each iteration, it computes from the already selected four-tuples the functions $\hat{Q}_1, \hat{Q}_2, \cdots, \hat{Q}_N$ and then selects as new four-tuple $(x_t^l, u_t^l, r_t^l, x_{t+1}^l)$ the one that maximizes $\epsilon(x_t^l, u_t^l)$. Our algorithm is limited to deterministic environments since we cannot from the sole knowledge of $\mathcal{F}$ compute the value of $\epsilon(x_t^l, u_t^l)$ when the environment is stochastic. The tabular version of the algorithm is given in Fig. 1 where the symbol f^j denotes a concise set containing j four-tuples.

(i) Set $j = 1$ and $f^j = \{ \underset{(x_t^l, u_t^l, r_t^l, x_{t+1}^l) \in \mathcal{F}}{\arg\max} |r_t^l| \}$

(ii) Compute $\hat{Q}_1, \hat{Q}_2, \cdots, \hat{Q}_N$ from f^j by using the fitted Q iteration algorithm

(iii) $f^{j+1} \leftarrow f^j \cup \{ \underset{(x_t^l, u_t^l, r_t^l, x_{t+1}^l) \in \mathcal{F} \setminus f^j}{\arg\max} (\underset{k \in \{1,2,\cdots,N\}}{\max} |\hat{Q}_k(x_t^l, u_t^l) - r_t^l - \gamma \underset{u \in U}{\max} \hat{Q}_{k-1}(x_{t+1}^l, u)|) \}$

(iv) If $j + 1 = n$ return f^{j+1} else $j \leftarrow j + 1$ and go back to step (ii).

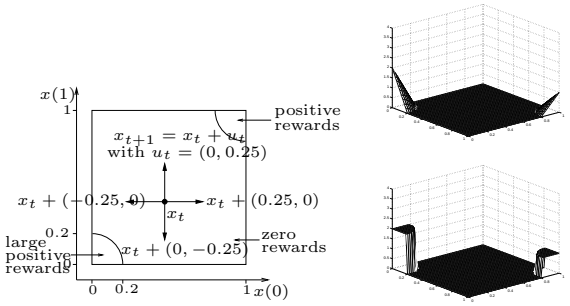
Figure 1: Concise set selection algorithm. The algorithm takes $\mathcal{F}$ as input and outputs a n element concise set.

Will such an algorithm identify concise sets ? The motivation for the algorithm lies in the expectation that by growing iteratively the concise set and by selecting at each iteration the four-tuple associated with the largest error, the resulting value of $\|\epsilon\|_\infty$ will be small. This in turn would lead to a tight bound (11) and, therefore, to a low suboptimality on the policy computed.

There are of course no guarantees that even if there exist concise sets leading to small values of $\|\epsilon\|_\infty$, our algorithm will be able to identify them. However, simulation results reported in the next section are rather encouraging.

We may also question whether it is a good strategy to identify concise sets that lead to a minimization of $\|\epsilon\|_\infty$. Indeed, we are interested in concise sets leading to policies with high expected returns and by identifying them through a criterion in which the expected return does not directly intervene, our algorithm may fall into several pitfalls. For example, we will show in the next section that when the environment is non-smooth, there may not exist sets of four-tuples for which $\|\epsilon\|_\infty$ drops below a certain value which leads the algorithm to select too many samples alongside the discontinuities.

Computational burdens. Our main motivation for selecting concise sets is to lighten the computational burdens of running the fitted Q iteration algorithm on the whole set of four-tuples. To analyze the potential computational benefits of our approach, two aspects need to be considered: the *computation time* and the *memory requirements*. Obviously, these two aspects strongly depend on the supervised learning method considered in the inner loop of the fitted Q iteration algorithm.



(a) System dynamics (b) Reward functions

Figure 2: The 2-dimensional maze problem.

Concerning the computation time, we should notice that the concise set selection approach requires to run the fitted Q iteration algorithm n times with sets of four-tuples ranging from 1 till n elements and that after each run, except the last one, the four-tuple that maximizes $\epsilon(x_t^l, u_l^l)$ has to be identified. Generally speaking, we can say that the computation time associated with such an approach grows rapidly with n and will only be lower than the one required to run fitted Q iteration on $\mathcal{F}$ if n is small compared to $\#\mathcal{F}$.

The memory gain realized by the concise set selection approach comes from the lower amount of memory required to use the supervised learning algorithm with n element training set rather than a $\#\mathcal{F}$ element training set. For some supervised learning methods like ensemble of regression trees which output a model whose size grows with the training set, the gain may be significant.

4 Simulations results

In this section, we evaluate the performances, in terms of capability to identify concise sets of samples, of the algorithm discussed in Section 3.

Test problems. Experiments are carried out on the deterministic 2-dimensional maze problem. Description of the system dynamics is done in Fig. 2a.[1] An object whose coordinates are $(x(1), x(2))$ travels in a square and can go either up, down, left or right. Zero rewards are always observed, except in the upper right corner of the square and in the bottom left corner. The decay factor γ is equal to 0.5.

Two different reward functions are considered in our experiments. One is continuous and the other discon-

[1] The exact system dynamics is given by the following equation:

$$x_{t+1}(i) = \begin{cases} x_t(i) + u_t(i) & \text{if } 0 \leq x_t(i) + u_t(i) \leq 1 \\ 1 & \text{if } x_t(i) + u_t(i) > 1 \\ 0 & \text{if } x_t(i) + u_t(i) < 0 \end{cases} \qquad \forall i = \{1, 2\}.$$

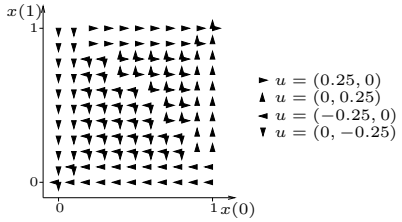


Figure 3: Representation of the optimal control stationary policy μ^* for 2-dimensional maze problem when the continuous reward function is considered.

tinuous (see Fig. 2). It can be shown that with the continuous (discontinuous) reward function, the corresponding Q_N-functions are continuous (discontinuous).

Figure 3 sketches the optimal stationary policy corresponding to the control problem with continuous reward function. To each point of the set $\{x \in X | \exists i, j \in \mathbb{N} | x = (0.1 * i, 0.1 * j)\}$, the figure draws control actions which are optimal ones.

Supervised learning method. In all our experiments, the supervised learning algorithm used inside fitted Q iteration is an ensemble of regression trees method named Extra-Trees.[2] This algorithm produces piecewise constant models. Each piece of the model is a hyperrectangle with each face parallel to all but one axes of the input space.

Value of N in fitted Q iteration. We have chosen to carry out ten iterations with the fitted Q iteration algorithm, i.e. $N = 10$. The policy chosen to approximate the optimal stationary policy is therefore $\hat{\mu}^*_{10}$. A larger value of N could at best lead to an upper bound (11) whose value is $\frac{2\gamma^N B_r}{(1-\gamma)^2} \simeq 0.008$ tighter.

Generation of the four-tuples. We consider in our experiments different sets $\mathcal{F}$. The mechanism that generates these different sets is the same. It considers one-step episodes with the initial state of each episode chosen at random in X and adopts as control policy a totally random policy.

Estimation of $\|\epsilon\|_\infty$ and $\|J^{\mu^*} - J^{\hat{\mu}^*_N}\|_\infty$. To estimate $\|\epsilon\|_\infty$ we compute $\max_{X_{test} \times U} \epsilon(x, u)$ where $X_{test} = \{x \in X | \exists i, j \in \mathbb{N} | x = (0.02 * i, 0.02 * j)\}$. Similarly the estimation of $\|J^{\mu^*} - J^{\hat{\mu}^*_N}\|_\infty$ is done by computing $\max_{X_{test}}(J^{\mu^*}(x) - J^{\hat{\mu}^*_N}(x))$.

[2]Description of the Extra-Trees algorithm may be found in [2]. Three parameters are associated to this algorithm: the number M of trees to build, the number K of candidate tests at each node and the minimum number of elements to split a leaf n_{min}. These parameters values are: $M = 50$, K is chosen equal to dimension of the input space which is equal to 4 (X is a 2-dimensional space and u is described by a pair of values), $n_{min} = 2$ (the trees are fully developed).

Random sets. To assess performances of the concise sets f^n, we compare them with those of sets which are randomly and uniformly chosen from $\mathcal{F}$. These sets are referred to as random sets.

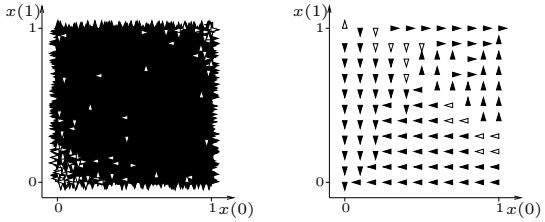
About the figures drawn. We explain how to interpret some of the figures drawn hereafter:

- *set of four-tuples*: several figures (Fig. 4a, Fig 6a and Fig 6d) draw sets of four-tuples. A four-tuple $(x^l_t, u^l_t, r^l_t, x^l_{t+1})$ is represented by a triangle on the $(x(1), x(2))$ plane with the center of the triangle being located at x^l_t and the orientation of the triangle giving information about u^l_t with the same convention as the one adopted in Fig. 3. Since the optimal control problem is deterministic, r^l_t and x^l_{t+1} can be deduced from x^l_t and u^l_t. A white triangle represents an element of the concise set f^n while a black triangle represents an element of $\mathcal{F} \setminus f^n$.
- *policy* $\hat{\mu}^*_{10}$: Figures 4b, 6b and 6e represent policies $\hat{\mu}^*_{10}$ computed in various conditions. Orientation of the triangles gives information about $\hat{\mu}^*_N(x)$ with the same convention as the one adopted in Fig. 3. If for a state x, the color of the triangle is white, $\hat{\mu}^*_{10}(x) \neq \mu^*(x)$. If it is black, $\hat{\mu}^*_{10}(x) = \mu^*(x)$.
- *function* $\max_{u \in U} \epsilon(x, u)$: Figures 5, 6c and 6f represent the function $\max_{u \in U} \epsilon(x, u)$. These figures give information about areas of the state space where the error function ϵ is the highest.

4.1 Continuous Q_N-functions

We consider here the case where the reward function is continuous. We first use our algorithm to select a 100 element concise set from a $10,000$ element set. The set f^{100} together with the set $\mathcal{F} \setminus f^{100}$ are represented on Fig. 4a. As we observe, many of the four-tuples are selected in areas of the state space where the reward function is different from zero. If fitted Q iteration takes f^{100} as input, it produces the policy $\hat{\mu}^*_{10}$ represented on Fig. 4b. To assess performances of this concise set, we have compared them with those obtained by 100 element random sets. For the concise set, the values of $\|\epsilon\|_\infty$ and $\|J^{\mu^*} - J^{\hat{\mu}^*_{10}}\|_\infty$ estimated are respectively equal to 0.490 and 0.399 while they are equal in average to 1.871 and 1.823 for the random sets. It is clear that our concise set was indeed able to lead to a better solution than random sets. It should however be noticed that if fitted Q iteration had been combined with $\mathcal{F}$, it would have lead to better values: 0.409 for$\|\epsilon\|_\infty$ and 0.199 for $\|J^{\mu^*} - J^{\hat{\mu}^*_{10}}\|_\infty$.

Table 1 further assesses performances of our algorithm. In the upper part of the table, we report performances of fitted Q iteration when combined with



(a) (b)

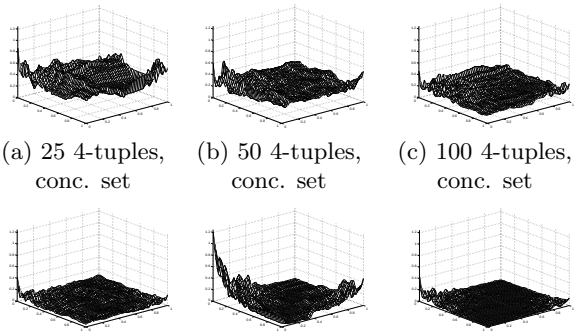
Figure 4: Figure a draws f^{100} and $\mathcal{F} \setminus f^{100}$. Figure b draws $\hat{\mu}^*_{10}$ when f^{100} is used with fitted Q iteration.

some concise sets of various sizes while in the lower part, average performances of several random sets as well as performances of $\mathcal{F}$ are given. The third column of the table estimates the value of $\|\epsilon\|_\infty$ and the fourth column the value of $\|J^{\mu^*} - J^{\hat{\mu}^*_{10}}\|_\infty$. By analyzing the content of these columns, we observe that concise sets lead to much better performances than random sets. In particular, performances of a 100 element concise set are better than average performances achieved by 2000 element random sets. These two columns also show that the upper bound (11) on the suboptimality of $\hat{\mu}^*_N$ is particularly loose. For example, if we suppose that estimates of $\|\epsilon\|_\infty$ and $\|J^{\mu^*} - J^{\hat{\mu}^*_{10}}\|_\infty$ are correct, the upper bound on the suboptimality of the policy is equal to $\frac{2(0.5)^{10}2+0.409(2-2(0.5)^{10})}{(1-0.5)^2} \simeq 1.6$ when $10,000$ four-tuples are considered as input of fitted Q iteration while the actual suboptimality of the policy is 0.199. Table 1 reports also the maximum value reached by the error functions over the different elements (x^l_t, u^l_t). Contrary to the values reported in columns three and four, these values can be computed from the sole knowledge of $\mathcal{F}$. They give to the reinforcement learning agent a lower bound on $\|\epsilon\|_\infty$. Remark that when the $10,000$ four-tuples are taken to compute $\hat{Q}_1$, $\hat{Q}_2$, $\cdots$, $\hat{Q}_{10}$, this value drops to zero, which is due to the fact that since the regression trees built are fully developed, they perfectly represent every training set of the sequence.

On Figs 5a-d we have drawn the function $\max_{u \in U} \epsilon(x,u)$ for increasing sizes of f^n. We observe that our algorithm tends to produce an error function constant over the state space and that by increasing the size of the concise set, the error function tends to decrease uniformly everywhere. It is obvious that this uniform decrease of the error would not have happened if the different (x^l_t, u^l_t) were not covering every area of the state-action space. These four figures should be put in comparison with Fig. 5e that draws $\max_{u \in U} \epsilon(x,u)$ when a 500 element random set is considered. By not

	size set	$\max_{\mathcal{F}} \epsilon(x^l_t, u^l_t)$	$\max_{X_{test} \times U} \epsilon(x,u)$	$\max_{X_{test}} (J^{\mu^*} - J^{\hat{\mu}^*_{10}})$
Concise sets	50	0.510	0.617	0.599
	100	0.348	0.490	0.399
	200	0.245	0.438	0.399
	500	0.117	0.439	0.199
	1000	0.073	0.450	0.199
Random sets (average values)	50	1.791	1.977	1.939
	100	1.652	1.871	1.823
	200	1.489	1.707	1.625
	500	1.062	1.259	1.330
	1000	0.766	1.015	0.903
	2000	0.629	0.818	0.679
$\mathcal{F}$	10000	0.	0.409	0.199

Table 1: Performances of different sets.



(a) 25 4-tuples, conc. set (b) 50 4-tuples, conc. set (c) 100 4-tuples, conc. set

(d) 500 4-tuples, conc. set (e) 500 4-tuples, rand. set (f) 10,000 4-tuples $= \mathcal{F}$

Figure 5: Function $\max_{u \in U} \epsilon(x,u)$.

selecting four-tuples in areas where the error is the highest, the error function reaches large values in the lower left and upper right corner. These values are even larger than those obtained by considering a 25 element concise set.

4.2 Discontinuous Q_N-functions

We consider here the case where the reward function is discontinuous which leads to some discontinuous Q_N-functions. First, we consider a set $\mathcal{F}$ composed of 500 elements and select from this 500 four-tuple set, a 100 element concise set. Figures 6a-c represent the results obtained. One should note that the error function reaches large values especially in the state space areas close to $(0,0)$ where the reward function is discontinuous. The circular aspect of the discontinuities together with the fact that the supervised learning algorithm produces piecewise constant models with each piece being an hyperrectangle, will always lead to a high value of $\|\epsilon\|_\infty$, what-

ever the finite set of four-tuples considered inside the fitted Q iteration algorithm is. In particular, since $\|\epsilon\|_\infty \geq \|\hat{Q}_1 - r(x,u)\|_\infty$ and since the height of largest discontinuity for the reward function is 2, $\|\epsilon\|_\infty$ will always at least be greater than 1.

Our algorithm selects four-tuples in areas of the state space where the error function is the highest. If when $\#\mathcal{F} = 500$, the concise set still contains four-tuples located in the upper right part of the square while the error function is greater in the lower left part, it is because $\mathcal{F}$ contains less than 100 four-tuples located in this lower left part.

Figures 6d-f gather simulation results when $\#\mathcal{F} = 100,000$. We observe now that almost all the four-tuples of the concise set are located alongside the largest discontinuity of the reward function and that the quality of the resulting policy is poor. In particular, since none of the elements of f^{100} gives information about the positive rewards that may be obtained in the upper right part of the state space, the policy computed tries mainly to drive the point to the lower left part of the state space.

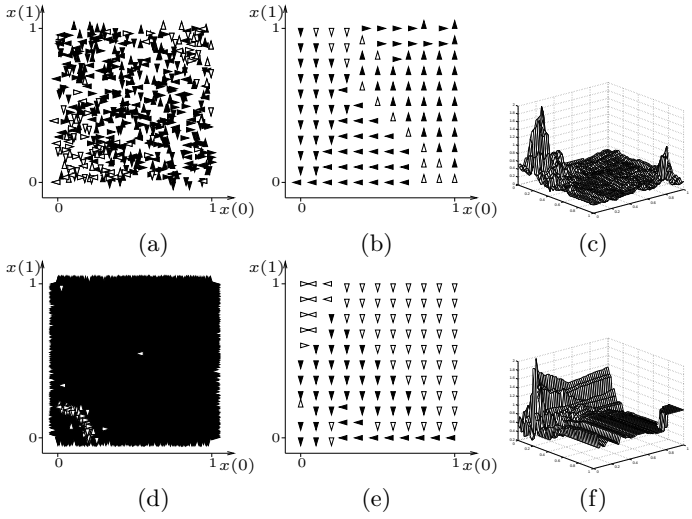


Figure 6: Figures a, b and c gather simulation results when $\#\mathcal{F} = 500$. Figure a sketches f^{100} and $\mathcal{F} \setminus f^{100}$, figure b the policy $\hat{\mu}^*_{10}$ and figure c $\max_{u \in U} \epsilon(x,u)$. Figures d, e and f reproduce similar results when $\#\mathcal{F} = 100,000$.

5 Conclusions and future work

In this paper, we have proposed an algorithm to identify concise sets of samples for a reinforcement learning agent interacting with a deterministic environment. This algorithm grows the concise sets iteratively. At each iteration, it computes a sequence of $\hat{Q}_N$-functions from the already selected samples by using the fitted Q iteration algorithm, associates an error function to these $\hat{Q}_N$-functions and selects the sample for which this error function takes its largest value. We showed through simulations that this algorithm has indeed the potential to identify good concise sets when the environment is smooth. However, we also found out that it may run into difficulties when non-smooth environments are considered. Indeed, in such cases the algorithm may select too many samples alongside the discontinuities.

Our primary motivation for selecting concise sets of samples was to lighten the computational burdens of running fitted Q iteration on the whole set of samples. However, we found out that the computation time associated with our proposed algorithm grows rapidly with the size of the concise set and may become larger than the time needed to run fitted Q iteration on the whole set of samples. We therefore suggest as future research direction to design faster versions of our algorithm.

Acknowledgments

Damien Ernst gratefully acknowledges the financial support of the Belgian National Fund of Scientific Research (FNRS).

References

[1] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, April 2005.

[2] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. Submitted.

[3] L. Grüne and W. Semmler. Using dynamic programming with adaptive grid scheme for optimal control problems in economics. *Journal of Economics Dynamics and Control*, 28:2427–2456, 2004.

[4] D.D. Lewis and J. Catlett. Heterogeonous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 148–156, San Francisco, CA, 1994. Morgan Kaufman.

[5] R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49:291–323, 2002.

[6] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2-3):161–178, 2002.

[7] M. Plutowski and H. White. Selecting concise training sets from clean data. *IEEE Transactions on Neural Networks*, 4(2):305–318, March 1993.