International Journal of Emerging Electric Power Systems

Volume 3, *Issue* 1 2005 *Article* 1066

Approximate Value Iteration in the Reinforcement Learning Context. Application to Electrical Power System Control.

Damien Ernst* Mevludin Glavic†

Pierre Geurts‡ Louis Wehenkel**

*University of Liège, ernst@montefiore.ulg.ac.be

†University of Liège, glavic@montefiore.ulg.ac.be

‡University of Liège, geurts@montefiore.ulg.ac.be

**University of Liège, lwh@montefiore.ulg.ac.be

Approximate Value Iteration in the Reinforcement Learning Context. Application to Electrical Power System Control.*

Damien Ernst, Mevludin Glavic, Pierre Geurts, and Louis Wehenkel

Abstract

In this paper we explain how to design intelligent agents able to process the information acquired from interaction with a system to learn a good control policy and show how the methodology can be applied to control some devices aimed to damp electrical power oscillations. The control problem is formalized as a discrete-time optimal control problem and the information acquired from interaction with the system is a set of samples, where each sample is composed of four elements: a state, the action taken while being in this state, the instantaneous reward observed and the successor state of the system. To process this information we consider reinforcement learning algorithms that determine an approximation of the so-called Q-function by mimicking the behavior of the value iteration algorithm. Simulations are first carried on a benchmark power system modeled with two state variables. Then we present a more complex case study on a four-machine power system where the reinforcement learning algorithm controls a Thyristor Controlled Series Capacitor (TCSC) aimed to damp power system oscillations.

KEYWORDS: reinforcement learning, power system control, electrical power oscillations damping, TCSC control, approximate value iteration

---

*D. Ernst, P. Geurts, M. Glavic and L. Wehenkel are with the Electrical Engineering and Computer Science Department, University of Liège, Sart-Tilman B28, 4000 Liège, BELGIUM. (e-mail : {ernst,geurts,glavic,lwh}@montefiore.ulg.ac.be). Damien Ernst and Pierre Geurts are postdoctoral researchers of the Belgian FNRS (Fonds National de la Recherche Scientifique), of which they kindly acknowledge the financial support.

1 Introduction

Power systems may be seen as large-scale systems whose dynamics are complex. Several authors have recognized the need for an intelligent and systematic learning method for power system control agents so that they can learn and update their decision making capabilities (Wehenkel, 1999; Taylor, 2000; Liu et al., 2000; Diu and Wehenkel, 2002). This need could be potentially met by a computational approach to learning known as reinforcement learning (RL) which aims at designing algorithms by which autonomous agents can learn to behave in some appropriate fashion, in some environment, from their interaction with this environment (see e.g. Kaelbling et al. (1996) for a broad overview).

The standard RL protocol considers an agent operating in discrete time, observing at time t the environment state x_t, taking an action u_t, and receiving back information from the environment in the form of the next state x_{t+1} reached and the instantaneous reward r_t obtained. The agent's objective is to learn as efficiently as possible from the set of four-tuples (x_t, u_t, r_t, x_{t+1}) acquired from interaction with the system a control policy which is expected to yield a maximal long term reward. In this context one of the main problems faced is how to exploit the information obtained from interacting with the environment in order to learn a good approximation of the optimal control policy.

If reinforcement learning algorithms have been applied successfully to a variety of practical problems including some famous examples such as Tesauro's TD-Gammon or Singh and Bertsekas's channel allocation algorithm (Tesauro, 1994; Singh and Bertsekas, 1997), many existing reinforcement learning algorithms fail to determine a good approximation of the optimal control policy when dealing with high-dimensional and/or continuous representations due to their poor ability to "generalize" to previously unseen data. In this paper we present a particular class of reinforcement learning algorithms that has revealed itself to be extremely efficient to generalize the information and that has therefore the potential ability to lead to successful applications when used to control large-scale systems like electric power systems. Furthermore, we present some encouraging simulation results obtained when these algorithms are used to control a Thyristor Controlled Series Capacitor (TCSC) aimed to damp electric power system oscillations.

This class of reinforcement learning algorithms performs well to generalize the information mainly because it can exploit the generalization capabilities of any supervised learning algorithm (ensemble of tree-based models, support-vector machines, etc.) to extract approximations of optimal control policies from the sets of four-tuples (x_t, u_t, r_t, x_{t+1}) gathered from interaction with the environment. Its principle is based on the Q-

learning idea[1] which consists of computing an approximation of the so-called Q-function from which the optimal policy can be determined in principle in a straightforward way.

To determine this Q-function, this class of RL algorithms solves a sequence of standard supervised learning problems. The training set for the first problem of the sequence is built from the sole knowledge of the set of four-tuples. For the Nth ($N > 1$) problem of the sequence, the training set is built by using the set of four-tuples in combination with the model produced at the $N-1$th iteration. This class of reinforcement learning algorithms has therefore been named *fitted Q iteration* so as to stress the fact that it allows to fit (using a set of four-tuples) in an iterative way any approximation architecture to the Q-function. The fitted Q iteration algorithm was introduced in Ernst et al. (2003) and studied carefully in Ernst et al. (2005) when used with tree-based methods. However, the idea of trying to approximate the Q-function from a set of four-tuples by solving a sequence of supervised learning problems may already be found in Ormoneit and Sen (2002) and is related to earlier work aimed to solve large-scale dynamics problem (Bellman et al., 1973; Tsitsiklis and Van Roy, 1996; Rust, 1997; Gordon, 1999). Furthermore, we will see later in this paper that this fitted Q iteration algorithm mimics in some sense the behavior of the well-known value iteration algorithm of the dynamic programming theory (Bellman, 1957).

The remainder of this paper is organized as follows. In Section 2, we formalize the problem of learning from interaction with the system and recall some classical results from optimal control theory upon which the reinforcement learning algorithms considered in this paper are based. In Section 3 we present the fitted Q iteration algorithm and discuss some simulation results obtained on a benchmark power system. Section 4 introduces two other families of reinforcement learning algorithms namely kernel-based and model-based ones that may be seen as particular cases of the fitted Q iteration algorithm. Section 5 discusses some computational issues the fitted Q iteration faces when it interacts in real time with a system and has to use at several instants the past information to provide a new estimate of the optimal control policy. Section 6 gathers some simulation results obtained when a reinforcement learning agent is used to control a Thyristor Controlled Series Capacitor aimed to damp electrical power oscillations. This section also discusses the strategy we have used

[1]The idea of learning the Q-function from interaction with a system has been introduced by Watkins (1989) and has been the dominant approach to reinforcement learning over the last fifteen years. There exist two main other families of reinforcement learning techniques. One searches directly for the optimal policy (Williams, 1992; Sutton et al., 2000). Algorithms of the other family are an intermediate between methods that learn the Q-function and methods that learn directly the optimal policy (Tsitsiklis and Van Roy, 2000).

to deal with the partial observability of the system. Finally, Section 7 discusses related work and Section 8 provides some concluding remarks. Two appendices compile relevant information about algorithms and the benchmark power system.

2 Problem formulation and dynamic programming

We consider a time-invariant stochastic system in discrete time for which a closed loop stationary control policy[2] must be chosen in order to maximize an expected discounted return over an infinite time horizon. We formulate hereafter the batch mode reinforcement learning problem in this context and we restate some classical results stemming from Bellman's dynamic programming approach to optimal control theory (introduced in Bellman, 1957) and from which the fitted Q iteration algorithm takes its roots.

2.1 Batch mode reinforcement learning 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 $P_w(w|x,u)$.[3]

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 supposed to be bounded by some constant B_r.

Let $\mu(\cdot) : X \rightarrow U$ 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:

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

[2]Indeed, in terms of optimality this restricted family of control policies is as good as the broader set of all non-anticipating (and possibly time-variant) control policies.

[3]In other words, the probability $P(w_t = w | x_t = x, u_t = u)$ of occurrence of $w_t = w$ given that the current state x_t and the current control u_t are x and u respectively, is equal to $P_w(w|x,u), \forall t = 0, 1, \cdots$.

where γ is a discount factor ($0 \leq \gamma < 1$) that weights 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 existence of an optimal stationary closed loop policy is a classical result from dynamic programming theory. It could be determined in principle by solving the Bellman equation (see below, Eqn (6)) given the knowledge of the system dynamics and reward function. However, the sole information that we assume available to solve the 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 four-tuple (x_t, u_t, r_t, x_{t+1}) of information. Since, except for very special conditions, it is not possible to determine exactly an optimal control policy from a finite sample of such transitions, we aim at computing an approximation of such a 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.

We do not make any particular assumptions on the way the set of four-tuples is generated. It could be generated by gathering the four-tuples corresponding to one single trajectory (or episode) as well as by considering several independently generated one or multi-step episodes.

We call this problem the *batch mode* reinforcement learning problem because the algorithm is allowed to use a set of transitions of arbitrary size to produce its control policy in a single step. In contrast, an *online* algorithm would produce a sequence of policies corresponding to a sequence of four-tuples.

2.2 Results from dynamic programming theory

For a temporal horizon of N steps, let us denote by

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

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

$$J_N^{\pi_N}(x) = \underset{\substack{w_t \\ t=0,1,\cdots,N-1}}{E} [\sum_{t=0}^{N-1} \gamma^t r(x_t, \pi_N(t, x_t), w_t) | x_0 = x] \tag{3}$$

its expected return over N steps. An N-step optimal policy π_N^* is a policy which among all possible such policies maximizes $J_N^{\pi_N}$ for any x. Notice that under mild conditions (see e.g. Hernández-Lerma and Lasserre, 1996,

for the detailed conditions) such a policy always does indeed exist although it is not necessarily unique.

Our algorithm exploits the following classical results from dynamic programming theory (Bellman, 1957):

1. The sequence of Q_N-functions defined on $X \times U$ by

$$Q_0(x,u) \equiv 0 \tag{4}$$
$$Q_N(x,u) = (HQ_{N-1})(x,u), \quad \forall N > 0, \tag{5}$$

converges (in infinity norm) to the Q-function, defined as the (unique) solution of the Bellman equation:

$$Q(x,u) = (HQ)(x,u) \tag{6}$$

where H is an operator mapping any function $K : X \times U \to \mathbb{R}$ and defined as follows:[4]

$$(HK)(x,u) = \underset{w}{E}[r(x,u,w) + \gamma \max_{u' \in U} K(f(x,u,w),u')]. \tag{7}$$

Uniqueness of solution of Eqn (6) as well as convergence of the sequence of Q_N-functions to this solution are direct consequences of the fixed point theorem and of the fact that H is a contraction mapping.

2. The sequence of policies defined by the two conditions[5]

$$\pi_N^*(0,x) = \arg\max_{u' \in U} Q_N(x,u'), \forall N > 0 \tag{8}$$
$$\pi_N^*(t+1,x) = \pi_{N-1}^*(t,x), \forall N > 1, t \in \{0,\ldots,N-2\} \tag{9}$$

are N-step optimal policies, and their expected returns over N steps are given by

$$J_N^{\pi_N^*}(x) = \max_{u \in U} Q_N(x,u).$$

3. A policy μ^* that satisfies

$$\mu^*(x) = \arg\max_{u \in U} Q(x,u) \tag{10}$$

is an optimal stationary policy for the infinite horizon case and the expected return of $\mu_N^*(x) \doteq \pi_N^*(0,x)$ converges to the expected return of μ^*:

$$\lim_{N \to \infty} J_\infty^{\mu_N^*}(x) = J_\infty^{\mu^*}(x) \quad \forall x \in X. \tag{11}$$

We have also $\lim_{N\to\infty} J_N^{\pi_N^*}(x) = J_\infty^{\mu^*}(x) \quad \forall x \in X$.

[4]The expectation is computed by using $P(w) = P_w(w|x,u)$.

[5]Actually this definition does not necessarily yield a unique policy, but any policy which satisfies these conditions is appropriate.

Equation (5) defines the so-called *value iteration algorithm*[6] providing a way to determine iteratively a sequence of functions converging to the Q-function and hence of policies whose return converges to that of an optimal stationary policy, assuming that the system dynamics, the reward function and the noise distribution are known. As we will see in the next section, it suggests also a way to determine approximations of these Q_N-functions and policies from a sample $\mathcal{F}$.

3 Fitted Q iteration algorithm

3.1 The algorithm

A tabular version of the fitted Q iteration algorithm is given in Figure 1. At each step this algorithm may use the full set of four-tuples gathered from observation of the system together with the function computed at the previous step to determine a new training set which is used by a supervised learning (regression) method to compute the next function of the sequence. It produces a sequence of $\hat{Q}_N$-functions, approximations of the Q_N-functions defined by Eqn (5).

Inputs: a set of four-tuples $\mathcal{F}$ and a regression algorithm.
Initialization:
Set N to 0 .
Let $\hat{Q}_N$ be a function equal to zero everywhere on $X \times U$.
Iterations:
Repeat until stopping conditions are reached

- $N \leftarrow N + 1$.
- Build the training set $\mathcal{TS} = \{(i^l, o^l),\, l = 1, \cdots, \#\mathcal{F}\}$ based on the the function $\hat{Q}_{N-1}$ and on the full set of four-tuples $\mathcal{F}$:

$$i^l = (x_t^l, u_t^l)\,, \tag{12}$$

$$o^l = r_t^l + \gamma \max_{u \in U} \hat{Q}_{N-1}(x_{t+1}^l, u)\,. \tag{13}$$

- Use the regression algorithm to induce from $\mathcal{TS}$ the function $\hat{Q}_N(x, u)$.

Figure 1: Fitted Q iteration algorithm

Notice that at the first iteration the fitted Q iteration algorithm is used

[6]Strictly, the term "value iteration" refers to the computation of the *value* function $J_\infty^{\mu^*}$ and corresponds to the iteration $J_N^{\pi_N^*} = \max_{u \in U} \underset{w}{E}[r(x, u, w) + \gamma J_{N-1}^{\pi_{N-1}^*}(f(x, u, w))], \forall N > 0$ rather than Eqn (5).

in order to produce an approximation of the expected reward $Q_1(x,u) = E_w[r(x,u,w)]$. Therefore, the considered training set uses input/output pairs (denoted (i^l, o^l)) where the inputs are the state-action pairs and the outputs the observed rewards. In the subsequent iterations, only the output values of these input/output pairs are updated using the value iteration based on the $\hat{Q}_N$-function produced at the preceding step and information about the reward and the successor state reached in each tuple.

It is important to realize that the successive calls to the supervised learning algorithm are totally independent. Hence, at each step it is possible to adapt the resolution (or complexity) of the learned model so as to reach the best bias/variance tradeoff at this step, given the available sample.

3.2 Algorithm motivation

To motivate the algorithm, let us first consider the deterministic case. In this case the system dynamics and the reward signal depend only on the state and action at time t. In other words we have $x_{t+1} = f(x_t, u_t)$ and $r_t = r(x_t, u_t)$ and Eqn (5) may be rewritten

$$Q_N(x,u) = r(x,u) + \gamma \max_{u' \in U} Q_{N-1}(f(x,u), u'). \tag{14}$$

If we suppose that the function Q_{N-1} is known, we can use this latter equation and the set of four-tuples $\mathcal{F}$ in order to determine the value of Q_N for the state-action pairs $(x_t^l, u_t^l), l = 1, 2, \cdots, \#\mathcal{F}$. We have indeed $Q_N(x_t^l, u_t^l) = r_t^l + \gamma \max_{u' \in U} Q_{N-1}(x_{t+1}^l, u')$, since $x_{t+1}^l = f(x_t^l, u_t^l)$ and $r_t^l = r(x_t^l, u_t^l)$.

We can thus build a training set $\mathcal{TS} = \{((x_t^l, u_t^l), Q_N(x_t^l, u_t^l)), l = 1, \cdots, \#\mathcal{F}\}$ and use a regression algorithm in order to generalize this information to any unseen state-action pair or, stated in another way, to *fit* a function approximator to this training set in order to get an approximation $\hat{Q}_N$ of Q_N over the whole state-action space. If we substitute $\hat{Q}_N$ for Q_N we can, by applying the same reasoning, determine iteratively $\hat{Q}_{N+1}$, $\hat{Q}_{N+2}$, etc.

In the stochastic case, the evaluation of the right hand side of Eqn (14) for some four-tuples (x_t, u_t, r_t, x_{t+1}) is no longer equal to $Q_N(x_t, u_t)$ but rather is the realization of a random variable whose expectation is $Q_N(x_t, u_t)$. Nevertheless, since a regression algorithm usually[7] seeks for an approximation of the conditional expectation of the output variable given the inputs, its application to the training set $\mathcal{TS}$ will still provide an approximation of $Q_N(x,u)$ over the whole state-action space.

[7] namely in the case of least squares regression, i.e. in the vast majority of regression methods.

3.3 Stopping conditions

The stopping conditions are required to decide at which iteration (i.e., for which value of N) the process can be stopped. A simple way to stop the process is to define a priori a maximum number of iterations. This can be done for example by noting that for a sequence of optimal policies μ_N^*, an error bound on the sub-optimality in terms of number of iterations is given by the following equation

$$\|J_\infty^{\mu_N^*} - J_\infty^{\mu^*}\|_\infty \leq 2\frac{\gamma^N B_r}{(1-\gamma)^2}. \tag{15}$$

Given the value of B_r and a desired level of accuracy, one can then fix the maximum number of iterations by computing the minimum value of N such that the right hand side of this equation is smaller than the tolerance fixed.[8]

Another possibility would be to stop the iterative process when the distance between $\hat{Q}_N$ and $\hat{Q}_{N-1}$ drops below a certain value. Unfortunately, for some supervised learning algorithms there is no guarantee that the sequence of $\hat{Q}_N$-functions actually converges and hence this kind of convergence criterion does not necessarily make sense in practice.

3.4 Control policy derivation

When the stopping conditions - whatever they are - are reached, the final control policy, seen as an approximation of the optimal stationary closed loop control policy is derived by

$$\hat{\mu}_N^*(x) = \arg\max_{u \in U} \hat{Q}_N(x,u). \tag{16}$$

3.5 Illustration: the OMIB optimal control problem

To illustrate the fitted Q iteration algorithm we consider a very simple power system referred to in the literature as the One Machine Infinite Bus (OMIB) power system and represented on Figure 2c. This system is composed of one generator connected to an infinite bus system through a transmission line. It has two state variables: the angle (δ) of the generator and its speed (ω). When the system is driven away from its equilibrium point, undamped electrical power (P_e) oscillations appear in the line. A variable reactance has been installed in series with the overall reactance

[8] Equation (15) gives an upper bound on the suboptimality of μ_N^* and not of $\hat{\mu}_N^*$. By exploiting this upper bound to determine a maximum number of iterations, we assume implicitly that $\hat{\mu}_N^*$ is a good approximation of μ_N^* (that $\|J_\infty^{\hat{\mu}_N^*} - J_\infty^{\mu_N^*}\|_\infty$ is small).

X_{system}. By controlling the value u of this variable reactance, it is possible to modify the electrical power transmitted in the line and damp the system.

The optimal control problem, precisely defined in Appendix B, has been stated in a way that the optimal stationary policy is indeed able to damp these electrical power oscillations. The state space for this optimal control problem is limited to the stability domain of the uncontrolled ($u = 0$) power system represented on Figure 2a plus a terminal state that is reached if the system exits from this domain. Policies that drive the system outside this stability domain are deemed to be unacceptable which is the reason why a very negative reward is associated to the exit of this domain. The action space has two values $u = -0.04$ and $u = 0$. More information about the physics of the OMIB power system control problem may be found in Pavella and Murthy (1994).

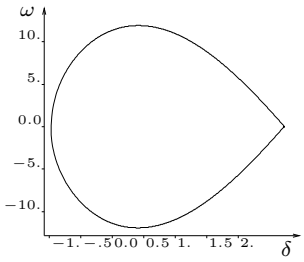
Four-tuples generation. To collect the four-tuples we have considered 10,000 one step episodes with x_0 and u_0 for each episode drawn at random in $X \times U$. In other words, we have repeated, after having initialized $\mathcal{F}$ to the empty set of four-tuples, $10,000$ times the following sequence of instructions

1. draw a state x_0 at random in X and initialize the system to this state
2. draw an action u_0 at random in U and apply this action to the system
3. observe r_0 and x_1
4. add (x_0, u_0, r_0, x_1) to the set of four-tuples $\mathcal{F}$.

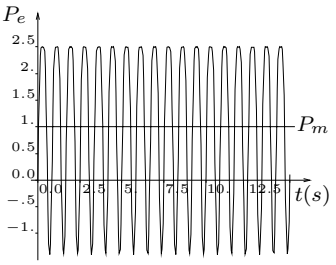
Simulation results. We have used here as supervised learning method a tree-based algorithm known as Extra-Trees. Its tabular version is given in Appendix A. This algorithm has been shown to perform particularly well on several benchmark problems (Ernst et al., 2005; Geurts et al., 2004). It has three parameters M (the number of trees that are built), K (the number of cut-directions considered at each node), and n_{min} (the minimum number of elements to split a node) which have been chosen here respectively equal to 50, the dimensionality of the input space (3), and 2.

The first iteration of the fitted Q iteration algorithm produces a function $\hat{Q}_1(x, u)$. To this function $\hat{Q}_1$ corresponds a policy $\hat{\mu}_1^*(x) = \max_{u \in U} \hat{Q}_1(x, u)$ that we have represented on Figure 3a. White bullets correspond to states x for which $\hat{\mu}_1^*(x) = 0$ (or equivalently states for which $\hat{Q}_1(x, 0) > \hat{Q}_1(x, -0.04)$) and black bullets states for which $\hat{\mu}_1^*(x) = -0.04$ (or equivalently states for which $\hat{Q}_1(x, 0) < \hat{Q}_1(x, -0.04)$). Successive policies $\hat{\mu}_N^*$ for increasing N are given on Figures 3b-3f.

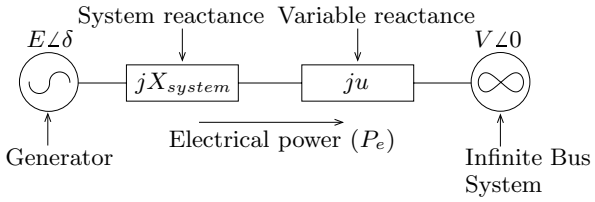
The trajectory obtained by simulating the system from $(\delta, \omega) = (0, 8)$ when controlled by policy $\hat{\mu}_{200}^*$ is represented on Figure 4a. The system



(a) Stability domain of the uncontrolled ($u = 0$) OMIB power system.

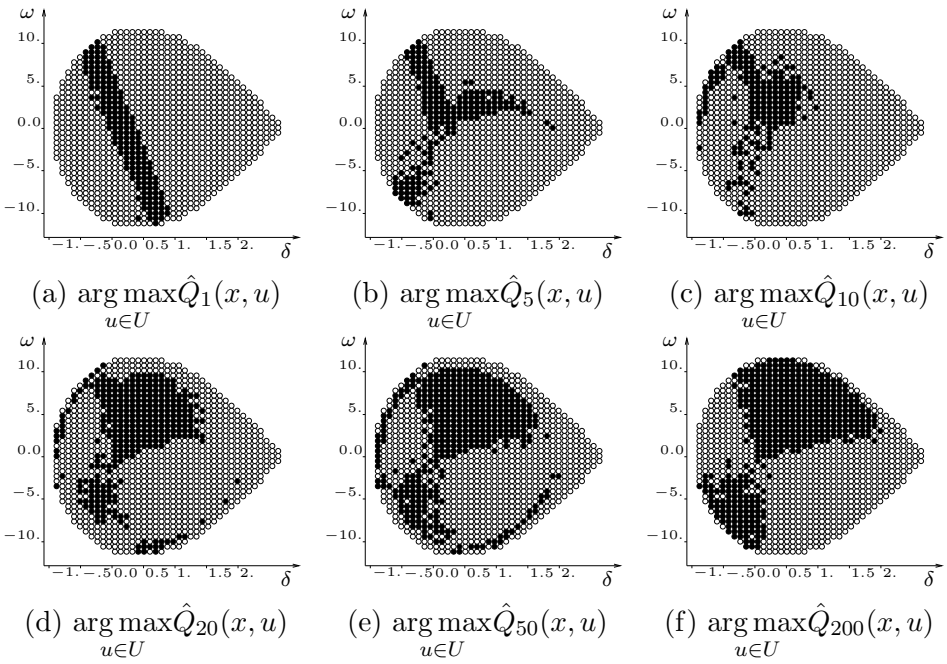


(b) Electrical power oscillations observed when at $t = 0$ $(\delta, \omega) = (0, 8)$ and $u = 0$ in the aftermath.



(c) Representation of the One Machine Infinite Bus System.

Figure 2: " The OMIB power system" optimal control problem.

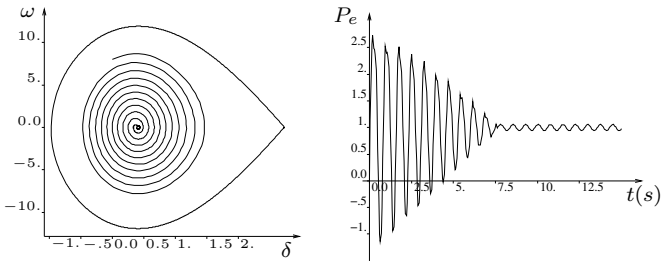


(a) $\arg\max_{u\in U}\hat{Q}_1(x,u)$ (b) $\arg\max_{u\in U}\hat{Q}_5(x,u)$ (c) $\arg\max_{u\in U}\hat{Q}_{10}(x,u)$

(d) $\arg\max_{u\in U}\hat{Q}_{20}(x,u)$ (e) $\arg\max_{u\in U}\hat{Q}_{50}(x,u)$ (f) $\arg\max_{u\in U}\hat{Q}_{200}(x,u)$

Figure 3: Representation of $\arg\max_{u\in U}\hat{Q}_N(x,u)$ for different values of N. The evaluation is carried out for the $(\delta, \omega) = (0.1 * i, 0.5 * j)$ with $i, j \in \mathbb{Z}$ that belong to X. Computation is done with a set composed of $10,000$ four-tuples (x_t, u_t, r_t, x_{t+1}) with (x_t, u_t) chosen at random in $X \times U$.

"gets closer" to the equilibrium point of the system, a sign that the electrical power oscillations are well damped which is confirmed by Figure 4b that represents the corresponding evolution of the electrical power.



(a) Trajectory (b) Electrical power

Figure 4: Trajectory and electrical power observed when at time $t = 0$ $(\delta, \omega) = (0, 8)$ and the policy $\arg\max_{u \in U} \hat{Q}_{200}(x, u)$ is used in the aftermath.

It is interesting to have a criterion able to assess the quality of each policy $\hat{\mu}^*_N$, that is to be able to determine how good a policy approximates the actual μ^*. The optimal stationary policy being here unknown, this criterion should not involve directly μ^*. Since an optimal stationary policy is a policy that maximizes the expected return for every initial state x_0, we measure the quality of policies using a pragmatic criterion based on the expected return and which *scores* a policy μ by the value of $J^\mu_\infty(x)$ averaged over a set of states X^i chosen *independently* from the set of four-tuples $\mathcal{F}$.

We have chosen here a set $X^i = \{(\delta, \omega) \in X | (\delta, \omega) \in X | \exists i, j \in \mathbb{Z} | (\delta, \omega) = (0.1 * i, 0.5 * j)\}$ and computed the score for each policy $\hat{\mu}^*_N$, $N \in \{1, \cdots, 200\}$. The result is represented on Figure 6a by the curve labeled "10,000 four-tuples". As one may observe, the score grows at the beginning rapidly with N to reach after a certain number of iterations an almost constant value.

Until now we have used a set $\mathcal{F}$ composed of $10,000$ four-tuples. We consider now two other sets of four-tuples generated in the same conditions but composed of less elements. One has 5000 four-tuples and the other 1000 four-tuples. As we may observe on Figure 6a, the score of the induced policy decreases when less four-tuples are used which is normal since the amount of information available to the fitted Q iteration algorithm to determine an approximation of the optimal stationary policy is smaller. Note that the score obtained by using 5000 four-tuples is only slightly lower than the score obtained with $10,000$ four-tuples. However, the score decreases significantly when only 1000 four-tuples are used.

4 Two particular cases: kernel-based reinforcement learning and model-based reinforcement learning

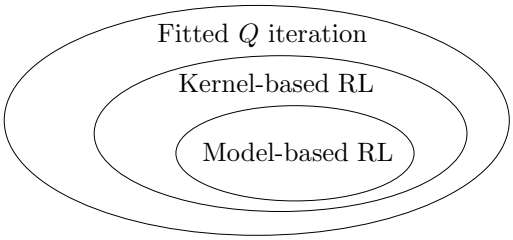


Figure 5: Approximate value iteration in the reinforcement learning context: relations between existing algorithms.

Two other classes of reinforcement learning algorithms directly based on the value iteration algorithm have been introduced in the RL literature. One is known as *kernel-based reinforcement learning* and is a particular case of the fitted Q iteration algorithm for which only kernel-based supervised learning methods are considered (Ormoneit and Sen, 2002; Ormoneit and Glynn, 2002). The other is known as *model-based reinforcement learning*. This latter class reconstructs from the set of four-tuples the structure of a Markov Decision Process (MDP) having a finite state space and action space, solves it and extends its solution to the original control problem (Sutton, 1990; Moore and Atkeson, 1993; Ernst, 2003). Actually, these model-based algorithms may be seen as a particular case of kernel-based reinforcement learning.

Figure 5 represents the relations that exist between these three classes of algorithms.

4.1 Kernel-based reinforcement learning

A kernel-based supervised learning method is a method that given a sample $\mathcal{TS} = \{(i^1, o^1), \ldots, (i^{\#\mathcal{TS}}, o^{\#\mathcal{TS}})\}$ produces at each call the model

$$f(i) = \sum_{l=1}^{\#\mathcal{TS}} k_{\mathcal{TS}}(i^l, i) * o^l, \tag{17}$$

with the kernel $k_{\mathcal{TS}}(i^l, i)$ being the same from one call to the other within the fitted Q iteration algorithm[9] and satisfying the normalizing condition:

$$\sum_{l=1}^{\#\mathcal{TS}} |k_{\mathcal{TS}}(i^l, i)| = 1, \forall i. \tag{18}$$

Supervised learning methods satisfying these conditions are for example the k-nearest-neighbors method, partition and multi-partition methods, locally weighted averaging, linear, and multi-linear interpolation. They are collectively referred to as kernel-based methods.

One important property of kernel-based reinforcement learning algorithms is that they guarantee the convergence of the sequence of $\hat{Q}_N$-functions, that is that they guarantee the existence of a function $\hat{Q} : X \times U \rightarrow \mathbb{R}$ such that $\forall \epsilon > 0$ there exists a $n \in \mathbb{N}$ such that:

$$\|\hat{Q}_N - \hat{Q}\|_\infty < \epsilon \quad \forall N > n.$$

To show it, one may first observe that the sequence of $\hat{Q}_N$-functions computed by the algorithm is determined by the recursive equation:

$$\hat{Q}_N(x,u) = \sum_{l=1}^{\#\mathcal{F}} k_{\mathcal{TS}}((x_t^l, u_t^l), (x,u))[r_t^l + \gamma \max_{u' \in U} \hat{Q}_{N-1}(x_{t+1}^l, u')], \tag{19}$$

$\forall N > 0$, with $\hat{Q}_0(x,u) = 0 \; \forall (x,u) \in X \times U$.

Then by rewriting Eqn (19):

$$\hat{Q}_N = \hat{H}\hat{Q}_{N-1} \tag{20}$$

where $\hat{H}$ is an operator mapping any function $K : X \times U \rightarrow \mathbb{R}$ defined through:

$$(\hat{H}K)(x,u) = \sum_{l=1}^{\#\mathcal{F}} k_{\mathcal{TS}}((x_t^l, u_t^l), (x,u))[r_t^l + \gamma \max_{u' \in U} K(x_{t+1}^l, u')] \tag{21}$$

and observing that this operator is a contraction on the Banach space of functions defined over $X \times U$ and the supremum norm, convergence follows.

We detail hereafter several classical supervised learning methods that may be seen as kernel-based supervised learning methods.

Partition based method. A partition based method considers one partition of the input space and determines a constant prediction in each

[9]This is true when the kernel does not depend on the output values of the training sample and when the supervised learning method is deterministic.

region of the partition by averaging the output values of the elements of the training set $\mathcal{TS}$ which belong to this region. Let $S(i)$ be the function that assigns to an input i the region of the partition it belongs to. A partition based method builds the model

$$f(i) = \sum_{l=1}^{\#\mathcal{TS}} \frac{I_{S(i)}(i^l)}{\sum_{j=1}^{\#\mathcal{TS}} I_{S(i)}(i^j)} * o^l \tag{22}$$

where $I_B(\cdot)$ denotes the characteristic function of the region B ($I_B(i) = 1$ if $i \in B$ and 0 otherwise) and its corresponding kernel $k_{\mathcal{TS}}$ is equal to:

$$k_{\mathcal{TS}}(i^l, i) = \frac{I_{S(i)}(i^l)}{\sum_{j=1}^{\#\mathcal{TS}} I_{S(i)}(i^j)}. \tag{23}$$

One particular case of the partition-based methods that we will consider later on is the so-called "grid-based methods", where the partition is obtained as the product of partitions of the elementary state and control variables.[10]

Multi-partition based method. A multi-partition based method considers several (p) partitions of the input space and builds a model for each partition by following the procedure used in the case of a single partition. The final model produced averages the predictions done by the different models. Let $S_m(i)$ be the function that assigns to each i the region of the mth partition it belongs to. A multi-partition based method produces the model

$$f(i) = \sum_{l=1}^{\#\mathcal{TS}} \frac{1}{p} \sum_{m=1}^{p} \frac{I_{S_m(i)}(i^l)}{\sum_{j=1}^{\#\mathcal{TS}} I_{S_m(i)}(i^j)} * o^l. \tag{24}$$

K-nearest-neighbors. The k-nearest-neighbors algorithm estimates the value of o for i by choosing the k values of i^l nearest the i for which one seeks an estimate, and by averaging their o^l values. It produces the model

$$f(i) = \sum_{l=1}^{\#\mathcal{TS}} \frac{1}{k} I_{nn_k(i,\mathcal{TS})}(i^l) * o^l, \tag{25}$$

where $nn_k(i, \mathcal{TS})$ is the set of the k nearest neighbors of i in $\mathcal{TS}$.

4.2 Model-based reinforcement learning

The main idea of model-based reinforcement learning algorithms is to determine from $\mathcal{F}$ an approximation of the system dynamics and of the

[10]Rather confusingly, some authors use the term "grid-based method" to denote what we have called the "partition-based methods". The grid-based methods are also called sometimes "histogram based methods".

reward function under the form of a finite Markov Decision Process. Once the MDP is determined, they derive an approximation of the Q-function by solving a Bellman equation.

More specifically, some of these methods partition the state space into $X_1, \cdots, X_m$ and the action space into $U_1, \cdots, U_n$ and consider a finite MDP having a state space $X^\delta = \{x_1^\delta, \cdots, x_m^\delta\}$ and an action space $U^\delta = \{u_1^\delta, \cdots, u_n^\delta\}$. The transition probabilities of the MDP are defined from the set of four-tuples by

$$p(x_j^i | x_i^\delta, u_h^\delta) = \frac{\sum_{l=1}^{\#\mathcal{F}} I_{U_h}(u_t^l) I_{X_i}(x_t^l) I_{X_j}(x_{t+1}^l)}{\sum_{l=1}^{\#\mathcal{F}} I_{U_h}(u_t^l) I_{X_i}(x_t^l)}, \quad (26)$$

$\forall i, j \in \{1, \cdots, m\}$, $\forall h \in \{1, \cdots, n\}$ and its reward function by:

$$r(x_i^\delta, u_h^\delta) = \frac{\sum_{l=1}^{\#\mathcal{F}} I_{U_h}(u_t^l) I_{X_i}(x_t^l) r_t^l}{\sum_{l=1}^{\#\mathcal{F}} I_{U_h}(u_t^l) I_{X_i}(x_t^l)} \quad (27)$$

$\forall i \in \{1, \cdots, m\}$, $\forall h \in \{1, \cdots, n\}$ and the approximate Q-function is determined through the following expression

$$\hat{Q}(x, u) = \sum_{i=1}^{m} \sum_{h=1}^{n} I_{X_i}(x) I_{U_h}(u) Q^\delta(x_i^\delta, u_h^\delta) \quad (28)$$

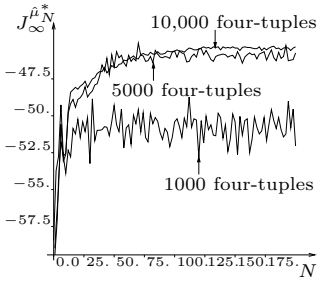
where the function Q^δ is the unique solution of the Bellman equation $Q^\delta = H^\delta Q^\delta$, H^δ being an operator that maps any function $K : X^\delta \times U^\delta \to \mathbb{R}$ onto

$$(H^\delta K)(x^\delta, u^\delta) = r^\delta(x^\delta, u^\delta) + \gamma \sum_{j=1}^{m} p^\delta(x_j^\delta | x^\delta, u^\delta) \max_{h \in \{1, \cdots, n\}} K(x_j^\delta, u_h^\delta). \quad (29)$$

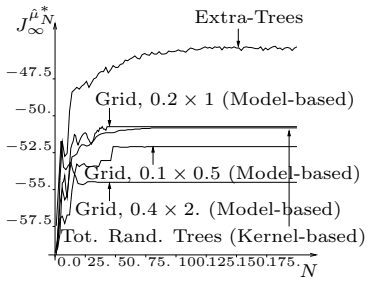
By rearranging these equations it can be shown that the function $\hat{Q}$ so computed is the unique solution of $\hat{Q} = \hat{H}\hat{Q}$ where $\hat{H}$ is the operator defined by Eqn (21) for which the kernel is determined by using a partition based method (Eqn (23)) where the $X \times U$ input space is partitioned into the $n * m$ subsets $X_i \times U_j$ $i \in \{1, \cdots, m\}$ $j \in \{1, \cdots, n\}$. They may therefore be considered as a particular type of kernel-based reinforcement learning.

4.3 Illustration: revisit of the OMIB control problem

In this section we illustrate the use of kernel-based and model-based reinforcement learning algorithms on the OMIB optimal control problem



(a) Influence of the set of four-tuples on the policy quality. Extra-Trees are used.



(b) Influence of the supervised learning method on the policy quality. $\#\mathcal{F} = 10,000$.

Figure 6: Score of $\hat{\mu}^*_N$ for different set of four-tuples and different supervised learning methods. The score is equal to $\frac{1}{\#X^i}\sum_{x\in X^i} J_\infty^{\hat{\mu}^*_N(x)}(x)$ with $X^i = \{(\delta,\omega) \in X | (\delta,\omega) \in X | \exists i,j \in \mathbb{Z} | (\delta,\omega) = (0.1 * i, 0.5 * j)\}$.

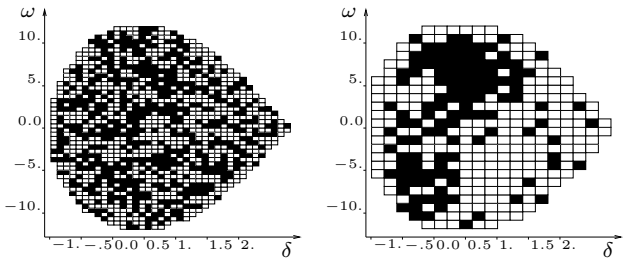
(Appendix B). The set of four-tuples used in combination with these algorithms is the $10,000$ element set described in Section 3.5.

As kernel-based supervised learning method we use a multi-partition based method. To build the partitions we use the first training set produced by the fitted Q iteration algorithm with the Totally Randomized Trees described in Appendix A with parameters $M = 50$ (i.e. fifty partitions are built) and $n_{min} = 2$. Previous tests on some benchmark problems have shown that by proceeding like this to build an ensemble of partitions, good results in terms of generalization of the information can be achieved (Ernst et al., 2005). The scores obtained by the policies $\hat{\mu}^*_N$ computed by the kernel-based reinforcement learning algorithm are represented on Figure 6b by the curve labeled "Tot. Rand. Trees (Kernel-based)". As we may observe, as N increases the score curve becomes a straight line which is normal since convergence to a unique policy is guaranteed by using a kernel-based RL method. This was not the case when using the Extra-Trees algorithm with the fitted Q iteration algorithm (see curve labeled "Extra-Trees"). However, with this latter technique better performances in terms of score were obtained than by using a kernel-based reinforcement learning method.

On Figure 6b we have also drawn scores obtained by using model-based reinforcement learning techniques when grid-based methods are used to partition the state space. Three different grids have been used. As illustrated on this figure, the grid with the tile size equal to 0.2×1 offers better results than the grid with bigger tile size (0.4×2) and than the grid with smaller time size (0.1×0.5).[11] Remark that even the best grid

[11]This observation can be explained by the bias-variance tradeoff. The error associated to a particular model can be decomposed into two terms, the bias and the variance.

provides much worse results that the Extra-Trees. Policies obtained by the grid with the 0.1×0.5 tiles and the 0.2×1 tiles have been represented respectively on Figures 7a and 7b. Black tiles represent areas of the state space where $\hat{\mu}^*$ is equal to -0.04 and white tiles areas where $\hat{\mu}^*$ is equal to 0. Remark that when the tiles are small, the control policy differs a lot from the one represented on Figure 3f. However, the policy corresponding to the grid that was providing the best results (0.2×0.1 tiles) shares more similarities with the one represented on Figure 3f.



(a) Tile size 0.1×0.5 (b) Tile size 0.2×1.

Figure 7: Policy obtained when using as supervised learning method regular grids. $\#\mathcal{F} = 10,000$.

5 Real-time learning

In previous sections we have introduced the fitted Q iteration algorithm that allows to extract from a set of four-tuples some knowledge about the optimal control policy of the system. We discuss in this section some computational issues this algorithm would face when it has to interact in real-time with an environment.

First, we define the notion of processing a stream of information in *real-time*, then we describe two main strategies that can be used in order to construct reinforcement algorithms based on the fitted Q iteration algorithm and which also have real-time performances.

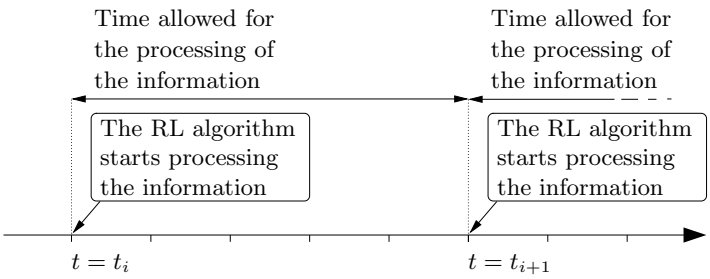


Figure 8: The problem of processing information in real-time.

5.1 Processing a stream of tuples in real-time

Figure 8 depicts graphically the idea of real-time processing. A system (or an environment) produces a stream of information on a tuple by tuple basis at successive (discrete) time instants. At time t, the system has produced t tuples

$$\mathcal{F} = \{(x_0, u_0, r_0, x_1), \cdots, (x_{t-1}, u_{t-1}, r_{t-1}, x_t)\},$$

and these are available to the learning algorithm to refresh the control policy used to drive the system, which is then used to determine the control signal for the next transition.

Intuitively, we would say that a learning algorithm is *real-time compatible with a discrete-time system* if the time it needs to improve the policy and produce the control signal at any time is smaller than the actual time-step used by the system to make its state-transitions. As suggested in Figure 8, this can be relaxed by allowing to run the learning algorithm at a slower rate than the discrete time imposed by the system transitions.[12] In that case, the learning algorithm processes the data not on a tuple by tuple basis but in packets of $t_{i+1} - t_i$ tuples. Notice also that in practice we require such an algorithm to use only a finite amount of memory.

5.2 Real-time fitted Q iteration algorithm

Since the time needed by the fitted Q iteration algorithm to produce a solution grows with the number of four-tuples, then the algorithm will not

When the model is not complex enough the error associated to the bias tends to be high while the error associated to the variance tends to be small. However, when the model is too complex (i.e. when too much parameters have to be learned), it is the opposite: the error associated to the bias tends to be small while the one associated to the variance tends to be high. The grid 0.2×1 seems therefore to offer here the best bias-variance tradeoff. More information about the bias-variance tradeoff may be found in Geurts (2002).

[12]However, we impose that the number of transitions, $t_{i+1} - t_i$, between successive runs of the learning algorithm remains bounded.

be real-time compatible if it is used as such.

We have identified two possible strategies in order to produce a real-time version of the fitted Q iteration algorithm. The first one limits the algorithm to some particular supervised learning methods for which the fitted Q iteration algorithm can be implemented in an incremental way. The other uses it in combination with another (real-time) algorithm that extracts from the stream of tuples a subset of tuples of bounded size.

Subclass of supervised learning methods. It is possible for some supervised learning algorithms to program the fitted Q iteration algorithm in such a way that part of the solution computed at time t_i can be used at time t_{i+1}, and so that the time required to compute a new solution can be bounded.

For example when the supervised learning method is a partition based method for which the state space and the action space are partitioned separately, it is achieved by observing that the solution produced by the fitted Q iteration algorithm is identical to the solution of a model-based reinforcement learning algorithm (Section 4) and by exploiting the fact that the MDP structure can be updated incrementally. Such an algorithm is described in Ernst (2003). Unfortunately the class of supervised learning methods that can be implemented in such a special way is, to our knowledge, limited to some particular partition and multi-partition based methods, methods which do not offer the best performances in terms of generalization of the information. Furthermore, these special implementations require partitioning (or multi-partitioning) the state-action space at the very beginning of the interaction process. Therefore, with such implementations, it is not possible to adapt the partitioning to the amount of information gathered in one area of the state-action space and to the shape of the Q-function. This may lead to a much too coarse or too fine partitioning in some areas and penalize strongly the results obtained. Such phenomena have been in some sense illustrated on Figure 6b (see also Section 4.3) where it is shown that too small grid tiles or too large grid tiles strongly decrease the quality of the solution obtained.

Identifying a relevant set of four-tuples. Another solution is at each t_i not only to compute the approximate optimal policy but also to select a limited number of four-tuples and removing the others from the set of four-tuples already collected.

One way of doing this, used by many real-time methods, is merely to use a First-In-First-Out (FIFO) queue of bounded size to keep the N most recent tuples. This strategy would also make sense in the context or adaptive control. Ernst (2005) explores the use of active learning algorithms to select the N *most informative four-tuples*. On several benchmark problems such a strategy has indeed shown significantly superior results when the system dynamics and the reward function were continuous.

6 A "real-world" example

In this section we lay out the procedure we have used to design an intelligent agent able to learn to control in an appropriate way a Thyristor Controlled Series Capacitor (TCSC) aimed to damp electrical power oscillations. First we explain the control problem considered. Then we discuss the way we have formulated this problem explicitely as a discrete-time optimal control problem and explain how to design, by using the material of previous sections, an agent potentially able to learn an appropriate way to control this TCSC while interacting in real-time with the system. Simulation results presented in this Section are taken from Ernst (2003) and have been also partially published in Ernst et al. (2004).

We discuss hereafter two different test cases. For the first case the agent is used to control the TCSC when the system operates in steady-state conditions while for the second case, the power system dynamics has been slightly modified in order to create self-sustained electrical power oscillations. Other simulations results concerning, among others, the ability of reinforcement learning agents to deal with oscillations that originate from short-circuits on the system or with time-varying loads are reported in Ernst (2003).

As we will see later, the system state will not be completely observable. The strategy we will use to cope with this problem will be to define a pseudo-state from the history of the observations done and the actions taken and proceed as if it was the real state of the system. The pseudo-state of the system at time t will be defined by an equation of the type:

$$\tilde{x}_t = (o_t, \cdots, o_{\max(0,t-Nbo+1)}, u_{t-1}, \cdots, u_{\max(0,t-Nbu)})$$

where o represents the observation done on the system, Nbo and Nbu determine respectively the number of successive observations and the number of successive actions taken by the RL algorithm that are used in the definition of the pseudo-state. In principle, the larger the values of Nbo and Nbu are, the better is the information about the system state contained in the pseudo-state. But increasing these numbers also increases the dimensionality of the (pseudo) state space and may therefore penalize the learning speed. More information about this strategy used to deal with the partial observability may be found in Kaelbling et al. (1998).

6.1 Presentation of the control problem

The four-machine power system used here is represented on Figure 9 and its characteristics are largely inspired from Kundur (1994). When this power system operates in steady-state conditions, the machines, identified by the symbols G1, G2, G3 and G4, produce approximately the same power: 700 MW and the two loads L7 and L9 consume respectively 990 and 1790 MW.

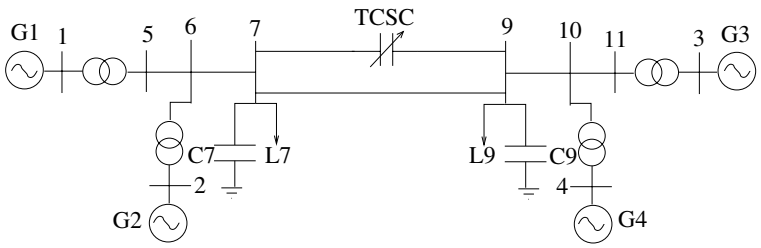


Figure 9: A four-machine power system.

This repartition of the active power production and consumption causes a power transfer of about 410 MW in the corridor connecting bus 7 to bus 9. The two lines that compose the corridor are such that the reactance of the one on which the TCSC is installed is twice as large as the other.

The loads are modeled according to the exponential model:

$$P = P_0(\frac{V}{V_0})^a \qquad Q = Q_0(\frac{V}{V_0})^b \tag{30}$$

where the subscript $_0$ identifies the values of the respective variables at the initial operating conditions (for example P_0 of the load L7 is equal to 990 MW).

Each machine of this power system is modeled in the same way with 15 state variables: 6 that correspond to the electrical model, 2 to the mechanical variables (the rotor angle and speed), 3 to the automatic voltage regulator (AVR) and 4 to the turbine (including the governor).

The TCSC can be considered as a variable reactance placed in series with a transmission line. The reactance of the TCSC, denoted by X_{TCSC}, responds to the first order differential equation:

$$\frac{dX_{TCSC}}{dt} = \frac{X_{ref} - X_{TCSC}}{T_{TCSC}} \tag{31}$$

where X_{ref} represents the TCSC reactance reference and where T_{TCSC} has been chosen, in accordance with the technical specifications of such a TCSC device equal to $60\,ms$ (Hingorani and Gyugyi, 2000; Ghandhari, 2000, see e.g.).

The control variable for this system is X_{ref} and is limited to a discrete set of values equal to $\{-61.57,\ -46.18,\ -30.78,\ -15.39,\ 0\}$. A value of $-61.57\,\Omega$ for X_{TCSC} corresponds approximately to a $30\,\%$ compensation of the line on which the TCSC is installed.

Our aim is to control this device to damp electrical power oscillations *by using only locally available measurements*.

6.2 Optimal control problem statement

In order to enable the proper operation of the reinforcement learning algorithm all the quantities used by the algorithm, that is the notion of states and rewards, must be defined on the basis of local measurements. We chose here a single local measurement, namely of the active power flow through the line in which the TCSC is installed.

This quantity is obtained at each time step of $50\,ms$. It is used to define the rewards and pseudo-states used by the RL algorithm (including the detection of loss of synchronism). To construct the pseudo-state that will be used inside the RL algorithm, we further need to define Nbo and Nbu (Eqn (30)). Preliminary simulations have shown that a choice $Nbo = 3$ and $Nbu = 2$ leads to a good compromise between information and convergence speed of the RL algorithm. Thus the pseudo-state at time t is defined by the following expression:

$$\tilde{x}_t = (P_{e_t}, P_{e_{t-1}}, P_{e_{t-2}}, u_{t-1}, u_{t-2}) \tag{32}$$

The aim of the control is to maximize damping of the electrical power oscillations in the line. This choice is motivated by the fact that damping improvement of these electrical power oscillations should also lead to an overall improvement of the power system damping. Thus, we define the reward by:

$$r(x,u) = \begin{cases} -|P_e - \overline{P}_e| & \text{if } |P_e| \leq 250\,\text{MW} \\ -1000 & \text{if } |P_e| > 250\,\text{MW} \end{cases} \tag{33}$$

where $\overline{P_e}$ represents the steady-state value of the electric power transmitted through the line, and the condition $|P_e| > 250\,\text{MW}$ is used to detect instability. When this latter condition is reached the learning and control algorithms stop interacting with the system until they are reinitialized. The discount factor γ is set to 0.98, which corresponds to a 90 % discount after about 5.5 seconds of real-time.

Note that the steady-state value of the electrical power is dependent on several aspects (operating point, steady-state value of X_{ref}) and so cannot be fixed before-hand. Thus, rather than to use a fixed value of $\overline{P}_e$, we estimate its value on-line using the following equation:

$$\overline{P}_e = \frac{1}{1200} \sum_{k=0}^{1199} P_{e_{t+1-k}}, \tag{34}$$

which is a moving average over the last $1200 * 50\,ms = 60\,s$.

6.3 The design of the reinforcement learning agent

Processing of the information. Since we want our agent to be able to process in real-time the information acquired from interaction with

Figure 10: The ϵ-Greedy policy

Input: Current state x of the system, $\hat{Q}(x,u)$ and ϵ
Algorithm:
Choose a number at random in the interval $[0,1]$ with a uniform probability.
If the number is inferior to ϵ then choose the control variable value randomly in U. Otherwise choose the control variable value randomly in $\{u \in U | \hat{Q}(x,u) = \max_{u' \in U} \hat{Q}(x,u')\}$.

---

the system (see Section 5) we have decided to use as supervised learning method a partition based method for which the (pseudo) state space and the action space are partitioned separately. The pseudo state space $[-250,250] \times [-250,250] \times [-250,250]$ is partitioned into $1,000,000$ of subsets, each subset being defined by an expression of the type

$$[-250+(i-1)\Delta P_e, -250+i\Delta P_e] \times [-250+(j-1)\Delta P_e, \\ -250+j\Delta P_e] \times [-250+(k-1)\Delta P_e, -250+k\Delta P_e]$$

where ΔP_e is equal to 5 MW and $i,j,k \in \{1,2,\cdots,100\}$. Since the action space is here finite and composed of a small number of elements ($\#U = 5$), we create a partition of the action space such that each element of this partition corresponds to a single element of the action space.

Concerning the elements of the (pseudo) state-action space partition that are not visited[13], we consider in the computational process that the value of $\hat{Q}_N(\tilde{x},u)$ for these elements is equal to 0 $\forall N \in \mathbb{N}$. Furthermore, the estimate of $\hat{\mu}^*$ is refreshed each time a new four-tuple is available.

Control policy used by the agent. The agent uses an ϵ-Greedy policy to control the system. With such a policy the agent selects with probability $1-\epsilon$ a greedy action (an action that maximizes $\arg\max_{u \in U} \hat{Q}(x,u)$) and with probability ϵ an action at random (see Figure 10 for a tabular version of the ϵ-Greedy policy). The smaller the value of ϵ, the better the agent exploits the approximate optimal control policy he has learned and the less he explores its environment. We will use in our simulations a constant and relatively small value of ϵ (0.01).

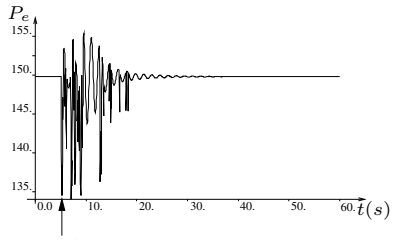


Figure 11: The system operates in steady-state conditions. The reinforcement learning agent takes control of the TCSC at $t = 5\,s$.

6.4 Control of the stable equilibrium point

We suppose that the TCSC is working like a fixed capacitor at full range of its capacity ($X_{TCSC} = -61.57$) and that the power system is in steady-state conditions. In such conditions the electric power transmitted through the TCSC is constant and equal to 149.81 MW. Then, at one instant, the agent enters into action to control the TCSC.

Notice that the best control strategy one could adopt would be to choose u always equal to -61.57 because by proceeding this way the electric power transmitted through the line would stay constant and the reward obtained would be maximal (indeed $P_{e_{t+1}}$ and $\overline{P_e}$ would both be equal to 149.81 MW and the reward r_t would be equal to 0).

But the agent we have designed adopts another strategy. Indeed, the first pseudo-state the agent observes is equal to

$$\tilde{x}_0 = (149.81, 149.81, 149.81, -61.57, -61.57).$$

Due to the fact that until non zero rewards are observed $\hat{Q}(x_0, u) = 0$, $\forall u \in U$, and that an ϵ-Greedy policy is being used, the value of u is chosen at random among U. Thus the agent will certainly in a first time drive the system away from its stable equilibrium point. The question is how far and how long it will drive the system away from this equilibrium point.

To answer this question, we have depicted on Figure 11 the temporal evolution of the electrical power in the line during the first 60 seconds. We see that, at time $t(s) = 5\,s$, when the agent begins to act, its interaction with the system indeed creates power oscillations. Nevertheless, quite quickly the algorithm is able to find out how to control the stable equilibrium point. Indeed after $20\,s$ these electric power oscillations have

[13]Suppose that $X_i \times U_j$ denotes an element of the state-action space partition. We say that $X_i \times U_j$ has been visited if among the set of four-tuples acquired from interaction with the system, there exists at least one four-tuple $(x_t^l, u_t^l, r_t^l, x_{t+1}^l)$ such that $x_t^l \in X_i$ and $u_t^l \in U_j$.

almost disappeared. The agent thus managed to find quite quickly a control strategy that allows the system to reach again the stable equilibrium point and stay in steady-state conditions.

6.5 Damping of self-sustained oscillations

We first consider the case where the TCSC is working like a fixed capacitor at full range of its capacity ($X_{TCSC} = -61.57$) but where the system dynamics has been (slightly) modified so that the initial stable equilibrium point becomes an unstable equilibrium point. This has been achieved by changing the parameters of the AVR of the machines G1, G2. Due to the unstable aspect of the equilibrium point, the system will be driven away from it and electric power oscillations will begin to grow. Saturation on the machines' field voltage will however limit the magnitude of these oscillations. Hence, after a certain time a stable limit cycle appears. The evolution of P_e over a period of $10\,s$ when the limit cycle has been reached is illustrated on Figure 12.

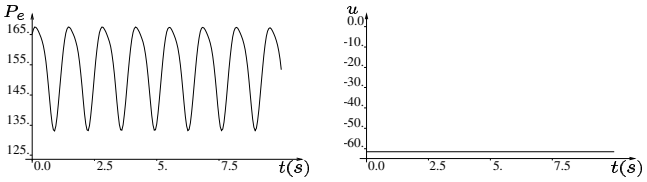
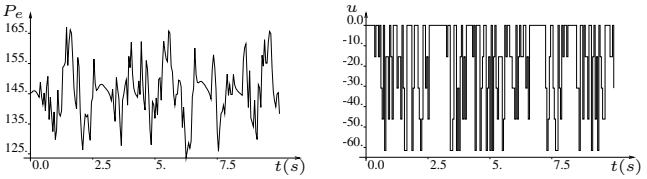


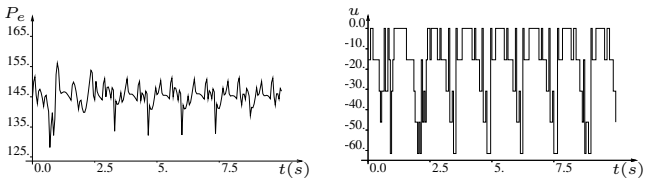
Figure 12: Electrical power oscillations (MW) occurring when u (Ω) is constant and equal to $-61.57\,\Omega$.



(a) (b)

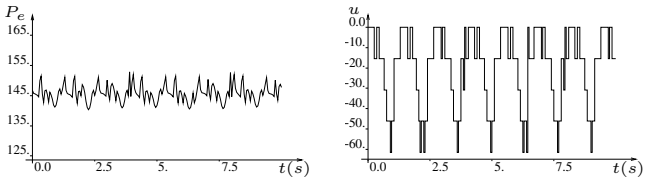
Figure 13: Electrical power (MW) and u (Ω) evolution after $10\,min$ of control.

Starting with this behavior, at a certain point in time, the agent enters into action to control the TCSC, in order to try to progressively reduce the amplitude of the limit cycle. For example, Figures 13a and 13b show the evolution of the electric power transmitted through the line (P_e) and the control action taken (i.e. the value of u) over a period of $10\,s$, after $10\,min$ of interaction with the system (the reinforcement learning agent



(a) (b)

Figure 14: Electrical power (MW) and u (Ω) evolution after $1\,h$ of control.



(a) (b)

Figure 15: Electrical power (MW) and u (Ω) evolution after $10\,h$ of control.

has imposed each $50\,ms$ the value of u for already $10\,min$). We observe that the magnitude of the P_e oscillations is still very large and that the evolution of the action u seems to be driven by an almost random process. The agent has not yet acquired enough information about the optimal policy to act efficiently.

After $1\,h$ of control (Figure 14a), however, the electric power transferred in the line starts being well damped. At the same time, a more organized structure appears in the sequence of control actions taken (Figure 14b).

After $10\,h$ of control (Figures 15a and 15b), the results are more impressive. The magnitude of the electrical power oscillations has strongly decreased. The variation of the control variable u has a periodic behavior of approximately the same frequency ($0.8\,Hz$) as the electrical power oscillations observed when no control occurs. The harsh aspect of the electrical power observed comes from the discontinuous variation of the control variable u.

7 Related work

The power system community started getting interested in application of RL methods to control power system quite recently. Different types of practical problems that may occur when using RL methods to control real power systems (curse of dimensionality, partial observability of the system states, and non-stationarity of the system), together with the strategies

that can be adopted to overcome them, were discussed in Ernst (2003); Ernst et al. (2004). Two RL modes are considered: the on-line mode and the off-line mode. In the on-line mode the RL agent interacts with the real system while in the off-line mode he interacts with a simulator. Two case studies made on a four-machine power system model are presented. The first one concerns the design of a dynamic brake controller tuned in off-line mode. The second concerns power oscillations damping by a TCSC device, in which the control agent uses adaptive on-line learning mode. The conceptual design of a hybrid multi-agent system for self-healing power infrastructure defense system presented in Liu et al. (2000) consists of three layers: reactive, coordination, and deliberative layer. The reactive layer (low-level layer) includes a set of heterogeneous agents acting locally over a particular set of power system components, plants or substations. The agents placed on the high-level layer, the deliberative layer, can analyze and monitor the power system from wide-area point of view. The coordination of a number of agents is an important issue. This task is envisioned to be assigned to the agents in the coordination layer. Further consideration on the robustness of the team of agents ended with the conclusion that the agents within the proposed system, in all three layers, need an intelligent and systematic learning method to learn and update their decision-making capability through direct interaction with the dynamic environment. The RL methods are mentioned as possible approach accompanied with a note that RL application within the proposed system should be done with great care, especially in the reactive layer, and intensive research in the field should be done prior its real application. The work from Liu et al. (2000) is further extended in Jung et al. (2002) as a feasibility study of a RL method for an agent's adaptive learning capability with load shedding control schemes. An investigation of a learning coordinated fuzzy-logic control strategy, based on the interconnected learning automata, for the control of dynamic quadrature boosters, installed distributively in a power system, to enhance power system stability is reported in Li and Wu (1999). A non model-based RL technique was employed to search for optimal fuzzy-logic controller parameters (the optimal fuzzy membership functions) according to a given performance index, to control the boosters in a coordinated fashion. The simulations, undertaken in the simple two-machine and thirty-machine power system, showed that the control algorithm can provide satisfactory performances, in a coordinated fashion, under different operation conditions with various disturbances. The use of a non model-based RL algorithm (temporal difference - TD) to optimize synchronous generator PID controller parameters was explored in Chan et al. (2000). A multi-agent based learning controller was evaluated on a three-machine power system and results showed that the proposed control scheme offers satisfactory learning performance and, following a fault disturbance, the learning controllers can damp out the multi-mode

oscillations of the power system rapidly. The adaptive heuristic critic was used as the basis for optimizing the control parameters through the use of an adaptive critic element and an adaptive search element. Application of RL methods to the control of a TCSC aimed to damp power system oscillations was proposed in Ernst and Wehenkel (2002). A detailed case study was carried out on a synthetic four-machine power system. The RL algorithms used, a non model-based (Q-learning) and a model-based (prioritized sweeping), were able to act correctly in order to damp power system oscillations with better results for the model-based algorithm. The difficulties to overcome in order to apply successfully rather general RL algorithms to TCSC control were discussed. These difficulties included notably the design of a function image of the oscillations damping quality and a strategy to cope with the only availability of local measurements (the only signal used by the controller was the electrical power transferred in the line, where the TCSC was included, measured at fixed intervals). The idea to employ a RL method (in on-line mode) in order to compute an approximation of the optimal control sequence comprising basic control laws derived from the concept of Control Lyapunov Functions has been introduced in Glavic et al. (2005). The results showed that proper combination of a stability oriented and a performance oriented (RL) control technique is a promising way to implement advanced control schemes in power systems. The design and practical implementation of optimal neurocontrollers that replace the conventional automatic voltage regulator and the turbine governor of turbogenerators on multimachine power systems have been presented in Venayagamoorthy et al. (2003). An adaptive critic design technique called dual heuristic programming has been used for this purpose. A comparison of a control strategy based on the fitted Q iteration algorithm with a classical controller (residue based damping controller of Rogers (2000)), both aimed to damp electro-mechanical oscillations in a synthetic four-machine power system, is given in Wehenkel et al. (2005). The results revealed that the RL based controller was able to outperform the classical one. In the same paper two control schemes were considered: one relying on local inputs only and another relying on local and remote signals with remote signals being chosen based on an observability analysis (signals corresponding to the best observability of the considered mode were chosen). Slightly better results were obtained when local and remote measurements were used to define the input signal.

8 Conclusions

In this paper we have considered a new type of reinforcement learning algorithm known as "fitted Q iteration" to design some intelligent agents for power system control. The main characteristic of this algorithm is to

reformulate the reinforcement learning problem as a sequence of standard supervised learning problems. It has the potential to address real-world power system control problems due to its great ability to generalize information. These good generalization performances are mainly due to the fact it can use in the context of reinforcement learning the generalization capabilities of any supervised learning methods and in particular some non-parametric methods like ensemble of regression trees methods.

However, we have shown that when an agent uses the fitted Q iteration algorithm to process in real-time the information he has acquired from interaction with the system, he may face some computational problems due to the fact that the computational burdens of the algorithm increase with the amount of information gathered. Two strategies to circumvent this problem have been discussed. The first one implements the fitted Q iteration algorithm in a certain way so that when new information is available, it can update incrementally the solution it has already computed. Unfortunately, these particular implementations only exist for a limited class of supervised learning methods that do not offer necessarily the best generalization performances. The second strategy uses the fitted Q iteration algorithm in combination with another algorithm that identifies the most relevant information. Encouraging simulation results obtained when using the first strategy to control in real-time a TCSC installed on a four-machine power system have been reported this paper. Some preliminary simulation results, not reported in this paper, have shown that when certain types of active learning algorithms are used to select the relevant four-tuples, the second strategy could lead to much better results but could also behave in a disastrous way, especially if the system dynamics and the reward function are not smooth enough.

In our opinion, state of the art reinforcement learning algorithms perform well enough to be used for the design of some intelligent agents for power system control. However, we recognize that these reinforcement learning algorithms may still behave poorly when used to solve certain types of control problems notably those for which the action space is large or those for which the system dynamics is significantly influenced by the learning process of other agents.

A Extra-Trees Induction Algorithm

The procedure used by the Extra-Trees algorithm to build a tree from a training set is described in Figure 16. This algorithm has two parameters: n_{min}, the minimum number of elements required to split a node and K, the maximum number of cut-directions evaluated at each node. If $K = 1$ then at each test node the cut-direction and the cut-point are chosen totally at random. If in addition the condition (iii) is dropped, then the tree

structure is completely independent of the output values found in the $\mathcal{TS}$, and the algorithm generates *Totally Randomized Trees*.
The score measure used is the relative variance reduction. In other words, if $\mathcal{TS}_l$ (resp. $\mathcal{TS}_r$) denotes the subset of cases from $\mathcal{TS}$ such that $[i_j < t]$ (resp. $[i_j \geq t]$), then the Score is defined as follows:

$$\text{Score}([i_j < t], \mathcal{TS}) = \frac{var(o|\mathcal{TS}) - \frac{\#\mathcal{TS}_l}{\#\mathcal{TS}} var(o|\mathcal{TS}_l) - \frac{\#\mathcal{TS}_r}{\#\mathcal{TS}} var(o|\mathcal{TS}_r)}{var(o|\mathcal{TS})}, \quad (35)$$

where $var(o|\mathcal{X})$ is the variance of the output o in the training set $\mathcal{X}$.

B Definition of the OMIB optimal control problem

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

$$\dot{\delta} = \omega \quad (36)$$

$$\dot{\omega} = \frac{P_m - P_e}{M} \quad \text{with} \quad P_e = \frac{EV}{u + X_{system}} \sin\delta \quad (37)$$

where P_m, M, E, V and X_{system} are parameters equal respectively to 1, 0.03183, 1, 1 and -0.4. P_m represents the mechanical power of the machine, M its inertia, E its terminal voltage, V the voltage of the terminal bus system and X_{system} the overall system reactance.
The discrete-time dynamics is obtained by discretizing the time with the time between t and $t+1$ chosen equal to $0.050\,s$. The value of u is chosen constant during each $0.050\,s$ interval.
If δ_{t+1} and ω_{t+1} are such that

$$\frac{1}{2}M\omega_{t+1}^2 - P_m\delta_{t+1} - \frac{EV}{X_{system}}\cos(\delta_{t+1}) > -0.438788$$

then a *terminal state* is supposed to be reached.[14]
State space: The state space X is composed of the elements (δ, ω) that satisfy[15]

$$\frac{1}{2}M\omega^2 - P_m\delta - \frac{EV}{X_{system}}\cos(\delta) \leq -0.438788 \quad (38)$$

and of a *terminal state*.
Action space: The action space $U = \{-0.04, 0\}$.

[14] A *terminal state* can be seen as a regular state in which the system is stuck and for which all the future rewards obtained in the aftermath are zero.

[15] These elements form the compact drawn on Figure 2a.

Build_a_tree($\mathcal{TS}$)
Input: a training set $\mathcal{TS}$
Output: a tree T;

- If

  (i) $\#\mathcal{TS} < n_{min}$, or

  (ii) all input variables are constant in $\mathcal{TS}$, or

  (iii) the output variable is constant over the $\mathcal{TS}$,

  return a leaf labeled by the average value $\frac{1}{\#\mathcal{TS}} \sum_l o^l$.

- Otherwise:
  1. Let $[i_j < t_j] =$ Find_a_test($\mathcal{TS}$).
  2. Split $\mathcal{TS}$ into $\mathcal{TS}_l$ and $\mathcal{TS}_r$ according to the test $[i_j < t]$.
  3. Build $T_l =$ Build_a_tree($\mathcal{TS}_l$) and $T_r =$ Build_a_tree($\mathcal{TS}_r$) from these subsets;
  4. Create a node with the test $[i_j < t_j]$, attach T_l and T_r as left and right subtrees of this node and return the resulting tree.

Find_a_test($\mathcal{TS}$)
Input: a training set $\mathcal{TS}$
Output: a test $[i_j < t_j]$:

1. Select K inputs,$\{i_1, ..., i_K\}$, at random, without replacement, among all (non constant) input variables.
2. For k going from 1 to K:
   (a) Compute the maximal and minimal value of i_k in $\mathcal{TS}$, denoted respectively $i^{\mathcal{TS}}_{k,min}$ and $i^{\mathcal{TS}}_{k,max}$.
   (b) Draw a discretization threshold t_k uniformly in $]i^{\mathcal{TS}}_{k,min}, i^{\mathcal{TS}}_{k,max}]$
   (c) Compute the score $S_k =$ Score$([i_k < t_k], \mathcal{TS})$
3. Return a test $[i_j < t_j]$ such that $S_j = \max_{k=1,...,K} S_k$.

Figure 16: Procedure used by the Extra-Trees algorithm to build a tree. The *Totally Randomized Trees* algorithm is obtained from this algorithm by setting $K = 1$ and by dropping the stopping condition (iii).

Reward function: The reward function is defined through the following expression:

$$r(x_t, u_t, w_t) = \begin{cases} -|P_{et+1} - P_m| & \text{if } x_{t+1} \neq \textit{terminal state} \\ -100 & \text{if } x_{t+1} = \textit{terminal state} \end{cases}$$

Decay factor: The decay factor γ has been chosen equal to 0.98.
Remarks:

- The dynamical system is integrated by using a trapezoidal method with a $0.01\,s$ integration time step.
- The uncontrolled system ($u = 0$) has one stable equilibrium point defined by:

$$(\delta_e, \omega_e) = (\arcsin \frac{X_{system} P_m}{EV}, 0) = (0.411, 0). \quad (39)$$

References

R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

R. Bellman, R. Kalaba, and B. Kotkin. Polynomial approximation - a new computational technique in dynamic programming: allocation processes. *Mathematical Computation*, 17:155–161, 1973.

K. H. Chan, L. Jiang, P. Tilloston, and Q. H. Wu. Reinforcement learning for the control of large-scale power systems. In *Proceedings of EIS'2000*, Paisley, UK, 2000.

A. Diu and L. Wehenkel. EXaMINE - Experimentation of a monitoring and control system for managing vulnerabilities of the European infrastructure for electrical power exchange. In *Proceedings of the IEEE PES Summer Meeting, panel session on power system security in the new market environment*, Chicago, USA, 2002.

D. Ernst. *Near Optimal Closed-Loop Control. Application to Electric Power Systems*. PhD thesis, University of Liège, Belgium, March 2003.

D. Ernst. Selecting concise sets of samples for a reinforcement learning agent. *Submitted*, 2005.

D. Ernst, P. Geurts, and L. Wehenkel. Iteratively extending time horizon reinforcement learning. In N. Lavra, L. Gamberger, and L. Todorovski, editors, *Proceedings of the 14th European Conference on Machine Learning*, pages 96–107, Dubrovnik, Croatia, September 2003. Springer-Verlag Heidelberg.

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

D. Ernst, M. Glavic, and L. Wehenkel. Power system stability control: reinforcement learning framework. *IEEE Transactions on Power Systems*, 19:427–435, February 2004.

D. Ernst and L. Wehenkel. FACTS devices controlled by means of reinforcement learning algorithms. In *Proceedings of the 14th Power System Computation Conference*, Sevilla, Spain, June 2002.

P. Geurts. *Contributions to Decision Tree Induction: Bias/Variance Tradeoff and Time Series Classification*. PhD thesis, University of Liège, Belgium, May 2002.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Submitted*, 2004.

M. Ghandhari. *Control Lyapunov Functions: A Control Strategy for Damping of Power Oscillations in Large Power Systems*. PhD thesis, Royal Instituate of Technology, Dept. of Electric Power Engineering, Electric Power Systems, 2000.

M. Glavic, D. Ernst, and L. Wehenkel. Combining a stability and a performance oriented control in power systems. *IEEE Transactions on Power Systems*, 20(1):525–525, 2005.

G.J. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Carnegie Mellon University, June 1999.

O. Hernández-Lerma and B. Lasserre. *Discrete-Time Markov Control Processes*. Springer, New-York, 1996.

N.G. Hingorani and L. Gyugyi. *Understanding FACTS*. IEEE press, 2000.

J. Jung, C.C. Liu, S.L. Tanimoto, and V. Vittal. Adaptation in load shedding under vulnerable operating conditions. *IEEE Transactions on Power Systems*, 17(4):1199–1205, 2002.

L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

P. Kundur. *Power System Stability and Control*. McGraw-Hill, 1994.

B. H. Li and Q. H. Wu. Learning coordinated fuzzy logic control of dynamic quadrature boosters in multimachine power systems. *IEE Part C-Generation, Transmission, and Distribution*, 146(6):577–585, 1999.

C.C. Liu, J. Jung, G.T. Heydt, and V. Vittal. The strategic power infrastructure defense (SPID) system. *IEEE Control System Magazine*, 20: 40–52, 2000.

A.W. Moore and C.G. Atkeson. Prioritized sweeping: reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

D. Ormoneit and P. Glynn. Kernel-based reinforcement learning in average-cost problems. *IEEE Transactions on Automatic Control*, 47 (10):1624–1636, 2002.

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

M. Pavella and P.G. Murthy. *Transient Stability of Power Systems: Theory and Practice*. John Wiley & Sons, 1994.

G. Rogers. *Power System Oscillations*. Kluwer Academic Publishers, 2000.

J. Rust. Using randomization to break the curse of dimensionality. *Econometrica*, 65(3):487–516, 1997.

S. Singh and D. Bertsekas. Reinforcement learning for dynamical channel allocation in cellular telephone systems. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 974–980. The MIT Press, 1997.

R.S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, San Mateo, CA, 1990. Morgan Kaufmann.

R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063, 2000.

C.W. Taylor. Response-based, feedforward wide-area control. Position paper for NSF/DOE/EPRI Sponsored Workshop on Future Research Directions for Complex Interactive Networks, Washington DC, USA, 16-17 November 2000., 2000.

G.J. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

J.N. Tsitsiklis and B. Van Roy. Feature-based methods for large-scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

J.N. Tsitsiklis and B. Van Roy. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12, 2000.

G.K. Venayagamoorthy, R.G. Harley, and D.C. Wunsch. Implementation of adaptive critic-based neurocontrollers for turbogenerators in a multimachine power system. *IEEE Transactions on Neural Networks*, 14(5): 1047–1064, 2003.

C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

L. Wehenkel. Emergency control and its strategies. In *Proceedings of the 13-th PSCC*, pages 35–48, Trondheim, Norway, 1999.

L. Wehenkel, M. Glavic, and D. Ernst. New developments in the application of automatic learning to power system control. In *Proceedings of the 15th Power System Computation Conference*, 2005.

R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.