DEA : An Architecture for Goal Planning and Classification

François Fleuret <francois.fleuret@inria.fr>
Eric Brunet <ericb@club-internet.fr>

April 18, 2000

Abstract

We introduce the Differential Efficiency Algorithm, which partitions a perceptive space during unsupervised learning into categories and uses them to solve goal-planning and classification problems. This algorithm is inspired by a biological model of the cortex proposing the cortical column as an elementary unit. We validate the generality of this approach by testing it on four problems with continuous time and no reinforcement signal until the goal is reached (constrained object moves, Hanoi tower problem, animat control, and simple character recognition).

1 Introduction

Two essential cognitive functions have generated multiple attempts at computer models. The first one, *classification*, corresponds to the ability to isolate relevant information among a set of perceptions, and to categorize it (objects or texture in images, words in sentences, etc.). The second one, *goal planning*, is the ability to generate sequences of actions to reach goal (prey hunting, path finding, strategy games, etc.) given a family of potential intermediate situations. Historically, those two problems have been studied separately. We will address specifically this issue in this paper.

The classification field can be decomposed into two main families of algorithms. The parametric models are designed to specific problems, while the non-parametric ones, like nearest neighbors, neural nets or decision trees (Ripley 1994) can be used to solve problems which have no *a priori* model. The latter seem to be closer to the natural capabilities of animal brains.

The planning problem has been studied from various points of view. Some of the models are deterministic and symbolic, like the SOAR architecture (Laird, Newell & Rosenbloom 1987). A few algorithms propose a connectionnist approach like neuronal models (Dehaene & Changeux 1997) or the SLUG architecture (Mozer & Bachrach 1991). Others are based upon a probabilistic representation of the problem within MDP theory (Howard 1960, Putterman 1994). They focus on the necessity to efficiently estimate the parameters of the model, given a decomposition of it into a set of situations (Barto, Sutton & Watkins 1989, Sutton 1990, Moore & Atkeson 1993, Kaelbling 1993). Some algorithms using MDP strategies repeatedly divide the situations to obtain a description with an appropriate resolution (Chapman & Kaelbling 1991, McCallum 1996*b*, Moore & Atkeson 1995).

We describe in this article the Differential Efficiency Algorithm, used to construct a planning network. This work is inspired by the biological model of the cortex proposed by Burnod (Burnod 1989). Formally, we show that, despite the connectionnist motivation, this model can be reduced to a standard MDP model, applied to a set of situations built iteratively. We show also that on particular problems, it is highly related to decision trees. It cumulates capabilities of both classification (binary trees) and goal planning (MDP) algorithms. There are several differences between the tasks we present in this paper, and what is usually addressed by goal-planning algorithms. First, we consider both discrete and continuous time problems. Second, there is no reward until the goal is reached ("goal of achievement" instead of "goal of maintenance"). Third, the choice of action is deterministic, thus we can estimate precisely if the policy chosen by the algorithm enables it to reach the goal or not.

The U-Tree algorithm proposed by McCallum (McCallum 1996*b*) already integrates perception and goal planning in the same structure, and splits the perception states in order to obtain a realistic Markovian model of the world. We will show in the results why the algorithm we propose is more efficient on the demanding tasks we try to solve.

In section 2 we describe our framework. In section 3, we formalize the problem and the algorithm in the framework of stochastic process theory. In section 4 we test an implementation of DEA on four different problems (constrained object moves, Hanoi tower problem, animat control, and character recognition) and compare it to the U-Tree algorithm.

2 Framework

2.1 Universe/agent decomposition

Since we want to develop a system to solve tasks in various environments, we distinguish two objects. The first one, the *universe*, describes an environment and a task to solve. The second one describes the resolution system and we will call it an *agent*. These two objects possess an internal state and interact through a channel called *interface* (this notion has already been proposed in (Booker, Goldberg & Holland 1989)). Using such a generic interface allows us to test DEA in various environments without changing it, and to validate its relative generality by estimating its performance the same way, whatever the task may be. Thus, we hope to limit, during the performance analysis, the bias due to a specific design of the agent in the problem.

2.2 Structure of a cortical model

Burnod raised the assumption that, given its neural connections, the anatomical unit called the cortical column could possess functional properties underlying the mechanisms of planning and categorization (Burnod 1989). The local processing of such units could result, at a global level, in invariant-recognition and sensorimotor sequence learning (Guigon, Grandguillaume, Otto, Boutkhil & Burnod 1994). The

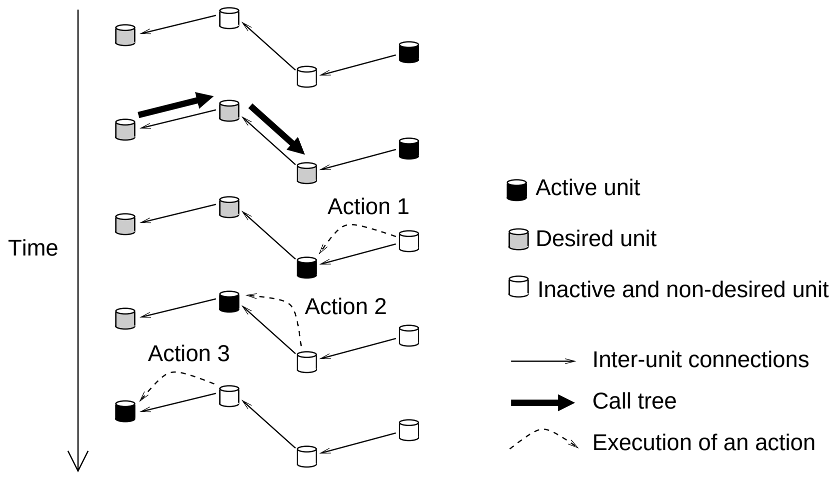


Figure 1: Call tree schema. The desire states propagate until they reach the neighborhood of a unit in the active state. Then, a succession of actions is performed.

cortical net processing described in those works relies on a fundamental planning mechanism.

The cortex is a net composed of *cortical units* linked to one another by inter-unit connections. Each of those units receives perceptive information and is able to trigger actions. It can recognize a set of perceptive states which corresponds to a situation of the animal in its environment. The unit is said to be in the *active state* if the animal is in the associated situation. Furthermore, a unit is said to be in the *desire state* if it is desirable for the animal to be in the associated situation. During learning, any connection between two units will be reinforced if there exists an elementary action that makes the animal reach the situation associated with the second unit starting from the situation associated with the first one, with high probability.

The planning is performed with *call trees*, each one being the propagation of a signal from units to units. The process starts by forcing the unit associated with the goal to be in the desire state. Then, the desire state spreads along strong inter-unit connections. This local rule of propagation is quite natural considering the meaning of the desire state and the semantic associated with the connections : if it is desirable to be in a given situation, it is desirable to be in a situation allowing to reach it. When the call tree reaches the neighborhood of an active unit, a chain of actions, corresponding to the successive connections along the call tree, is initiated. The whole process is represented in figure 1. We will see in 3.3.2 how the call-tree mechanism is related to probability estimations.

2.3 DEA and Planning Network

The algorithm we propose in this paper is inspired by the biologically plausible model described above. The formal similarity between the call-tree mechanism as we described it and certain estimations of probabilities in an MDP model leads us to use this probabilistic framework.

During a first step, the algorithm records the successive states of the universe while random actions are performed. The total number of actions required by such an exploration, even in a deterministic universe, can grow exponentially with the number of potential situations (a typical example of such a universe is the "Reset state space" (Koenig & Simmons 1996)). This problem can be partially solved by the restriction of this random exploration to a biased one (Kaelbling, Littman & Moore 1996). The usage of such "reflex" actions during the learning period is justified to some extent by the existence of such behavior in mammals and humans (Piaget 1964).

When this random step is over, one can estimate the strengths of the inter-unit connections by considering the record of the successive states of the universe. Using those estimations, the learning process consists in progressively splitting the units, and thus in increasing the accuracy of the universe model associated to the net. One can compare this mechanism with the U-tree algorithm (McCallum 1996*b*) or with the Parti-Game algorithm (Moore & Atkeson 1995).

3 Mathematical formalization

3.1 Universe definition

3.1.1 Formalization

We define a universe as a black box which possesses an internal time dependent state. One can control how this state evolves using *actions*, and can know partially its value through boolean projections called *parameters*. One of those parameters defines a subset of the state space called the *goal situation*. Formally, given two finite sets $\mathcal{A}$ and $\mathcal{P}$, the first one defining a set of actions, the second one a set of parameters, the following are the components of a universe :

- A set Υ, the internal state space ;
- A random variable $\upsilon_0 \in \Upsilon$, the initial state ;
- An element $g \in \mathcal{P}$, the goal situation ;
- A function $p : \Upsilon \rightarrow \{0,1\}^{\mathcal{P}}$, the value of the parameters ;
- A sequence of independent and identically distributed stochastic processes $T_t : \Upsilon \times \mathcal{A} \rightarrow \Upsilon$, the dynamic of the internal state.

For each time step t, depending on the internal state at time t and the action performed, the process T_t determines the state at time $t+1$. In all the universes that we will consider in this article the T_t are almost deterministic, and involve randomness only to re-initialize the state each time the goal is reached.

We will denote by $\mathcal{S} = \{0,1\}^{\mathcal{P}}$ the set of parameter value configurations. The set $\mathcal{I} = \mathcal{A} \times \mathcal{S}$ corresponds to the interface we described in 2.1. It is important to notice that the cardinality of $\mathcal{S}$ can be smaller than that of Υ, and therefore, the information transmitted by p can be incomplete. In the rest of the article, we will denote by υ_t the process associated with the internal state space, and $\sigma_t = p(\upsilon_t)$ the process associated with the parameter values.

3.1.2 Example

An example of a universe is presented figure 2. It is composed of a cursor that can move step by step along a horizontal segment, a hundredth of the segment length each time. The goal situation is reached when the cursor is in the center third of the segment, and the parameters indicate only which of the three thirds the cursor is in. So, we have :

- $\mathcal{A} = \{\triangleleft,\ \triangleright\}$; $\triangleleft$ and $\triangleright$ represent the moves to the left and the right ;
- $\mathcal{P} = \{t_1,\ t_2,\ t_3\}$; each parameter is associated with one third of the segment ;

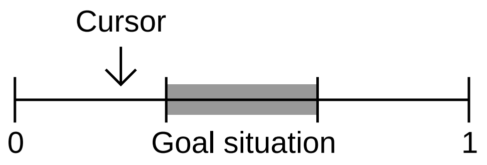


Figure 2: Universe example. The cursor can move to the right and the left.

- $g = t_2$ is the goal situation ;
- $\Upsilon = [0, 1]$; the internal state representing the location of the cursor ;
- $\upsilon_0 \sim \mathcal{U}(\Upsilon)$; initial state of the universe ;
- $p_{t_1} = 1_{[0,\frac{1}{3}]}$, $p_{t_2} = 1_{]\frac{1}{3},\frac{2}{3}]}$, $p_{t_3} = 1_{]\frac{2}{3},1]}$;
- The T_t are deterministic and defined by $\forall t,\ \forall \upsilon \in \Upsilon$,
  - $T_t(\upsilon, \triangleleft) = \max\{0, \upsilon - \frac{1}{100}\}$
  - $T_t(\upsilon, \triangleright) = \min\{1, \upsilon + \frac{1}{100}\}$

3.2 Definition of an agent

Given a universe, we define an agent as an abstract object, that chooses at each time step an action to perform depending on the actions done in the past, and on the past and current values of the parameters. Those actions have to be chosen to reach the goal situation starting from any situation. Formally, given an interface $\mathcal{A} \times \mathcal{S}$ and a parameter g representing the goal situation, an agent is a sequence of independent stochastic processes :

$$A_t : (\mathcal{S} \times \mathcal{A})^t \times \mathcal{S} \rightarrow \mathcal{A}$$

It is impossible to accurately formalize how efficiently we expect the agent to reach the goal situation. In most cases, a pure random strategy reaches almost surely the goal situation in a finite time. Thus, one has to take into account the time the agent needs to reach the goal. An exact formalization would involve its ability to survive, which does not concern us here. So, we will restrict ourselves to the following simple criterion :

The agent should be able, after a given number of learning steps, to reach the goal situation, starting from any situation, in a fixed maximum duration. This duration will be different for each universe and will be of the order of the minimum time required to reach the goal, starting from the worst situation.

One can remark that an agent defines the evolution of the state of the universe. Indeed, denoting by α_t the process associated with the actions, we have, $\forall t$,

$$\left\{ \begin{array}{rcl} \alpha_t & = & A_t(\sigma_0, \alpha_0, \ldots, \sigma_{t-1}, \alpha_{t-1}, \sigma_t) \quad \text{with} \quad \sigma_t = p(\upsilon_t) \\ \upsilon_{t+1} & = & T_t(\upsilon_t, \alpha_t) \end{array} \right. \tag{1}$$

3.3 Structure and functioning of the planning network

3.3.1 Unit activation definition

Given a universe, a planning network with N units is a family of functions $a_1, \ldots, a_N : \mathcal{S} \to \{0,1\}$. Each of them describes the activation of one unit, and we suppose in the rest of the article that the first one is associated with the goal situation, so that $\forall x \in \mathcal{S},\ a_1(x) = x_g$. We define a *policy* as a function $\pi : \{0,1\}^N \to \mathcal{A}$ which associates with each configuration of the activations of all the units an action to perform. The a_i and a policy π define an agent, and, according to (1), the process υ_t is well-defined. We will denote by $a_1(t), \ldots, a_N(t)$ the processes associated with the activations of the units, defined by $a_i(t) = a_i(p(\upsilon_t)),\ \forall i$. We assume that these processes are stationary.

According to the description of the planning network in 2.2, it seems quite natural to define the desired state d_i of a unit as the probability of reaching the goal situation if this unit were activated. The higher this probability, the more the organism has an interest in being in the situation associated with the unit. Formally, under the stationarity assumption, and letting P_π denote the probabilities corresponding to the usage of the policy π, we obtain :

$$d_i = \max_\pi \ P_\pi(\exists t < \infty,\ a_1(t) = 1 \mid a_i(0) = 1)$$

3.3.2 Estimation of the desire activations

We make the assumption in the rest of the article that the a_i are exclusive ; we will show we can force this property when we build the net :

$$\forall p,\ \sum_i a_i(p) = 1 \tag{2}$$

Any policy π can then be reduced to a function $\pi : \{1, \ldots, N\} \rightarrow \mathcal{A}$ which associates with each unit an action to perform. We can also define a process $c(t)$ which is the index of the activated unit at time t. We define t_0 as the time of the first activation modification : $t_0 = \min\{t > 0,\ c(t) \neq c(0)\}$. If we make a Markovian assumption[1] on c_t, we obtain the following standard equilibrium equation (Bellman 1957, Bertsekas & Tsitsiklis 1989) :

$$\left\{ \begin{array}{rcl} d_1 & = & 1 \\ \forall i \neq 1,\ d_i & = & \max_\alpha \sum_j \hat{P}_{\pi(i)=\alpha}(t_0 < \infty,\ a_j(t_0) = 1 \mid a_i(0) = 1)\ d_j \end{array} \right. \tag{3}$$

To estimate those probabilities we make the assumption that for any action $\alpha \in \mathcal{A}$ there exists a *maximum delay of effect*, $T_{max}(\alpha)$, the maximum duration required for an action to provoke a change. We also favor the shortest path by introducing a decreasing factor γ close to 1 (this term can be considered as the probability for the organism to survive during the next time step (Kaelbling et al. 1996)). So, (3) becomes :

$$\left\{ \begin{array}{rcl} d_1 & = & 1 \\ \forall i \neq 1,\ d_i & = & \gamma\ \max_\alpha \sum_j \hat{P}_{\pi(i)=\alpha}(t_0 < T_{max}(\alpha),\ a_j(t_0) = 1 \mid a_i(0) = 1)\ d_j \end{array} \right. \tag{4}$$

Let us define

$$\lambda^\alpha_{i,j} \ = \ \gamma\ \hat{P}_{\pi(i)=\alpha}(t_0 < T_{max}(\alpha),\ a_j(t_0) = 1 \mid a_i(0) = 1) \tag{5}$$

[1]We do not make the assumption that c_t is Markovian, but only that the process restricted to the instants of activation modifications is Markovian. This weaker assumption is satisfactory because the actions do not have an instantaneous effect.

Equation (4) then becomes :

$$\left\{\begin{array}{rcl} d_1 & = & 1 \\ \forall i \neq 1,\ d_i & = & \max_\alpha \ \sum_j \lambda^\alpha_{i,j}\ d_j \end{array}\right. \tag{DP}$$

And we have

$$\forall i,\ \pi(i) = arg\max_\alpha \ \sum_j \lambda^\alpha_{i,j}\ d_j \tag{AS}$$

To estimate the equilibrium of equation (DP), one should consider this equation as a recursive definition of a vector sequence, the limit of which is the equilibrium. If we consider the d_i to represent the desire activations, this iterative algorithm can be considered as the local call tree rule described in 2.2, the inter-cortical connections being represented by the $\lambda^\alpha_{i,j}$ coefficients. Likewise, the action selection rule is described by (AS). In some sense, it would be natural to consider the term we maximize in (AS) as an action desire activation : the most desired action becomes activated.

During the reflex period, the algorithm records all the values of the universe parameters and actions. Then, it can count the number of times a given event has occurred, and estimate the $\lambda^\alpha_{i,j}$ values as ratios. During the construction of the units as well, the algorithm uses only the record made during the reflex period ; the agent does not perform an active learning nor test a policy.

The estimation of the $\lambda^\alpha_{i,j}$ is not as simple as it is with discrete-time universes. In a continuous time universe, actions take a few cycles in order to produce an effect. Consequently, the efficiency of those actions wich are activated for very short periods of time are underestimated (this is similar to the truncated observations in the statistical analysis of clinical tries). Thus, the distribution of action durations induces a bias in the frequency of successful actions, and one has to use a correction factor to properly estimate the probability of transitions.

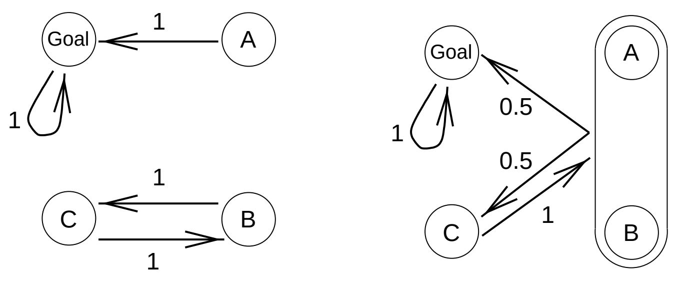


Figure 3: The Markovian process with four states shown on the left is poorly approximated by the process shown on the right, which mixes two of its states.

3.3.3 Weakness of the Markovian model

Even if the Markovian model used to justify the equation (3) is natural, it is pretty weak. It models the reality only if the a_i functions are appropriate. One can demonstrate the weakness of such a model with an example.

Let us consider the Markovian process with four states A, B, C, and $Goal$ shown on the left in figure 3. This process goes from state A to state $Goal$ with a probability 1, and if it is in one of the two other states, it oscillates between them forever. Let us now consider an approximation of this process that mixes the states A and B into one unique state $A \cup B$. The probability for this approximation to go from $A \cup B$ to C is equal to $P(B \mid A \cup B)$, and the probability to go to $Goal$ is $P(A \mid A \cup B)$. If those two probabilities are equal to 0.5, then the initial process is modeled by the process shown on the right in figure 3. Then, the estimated probability to reach the $Goal$ state in this rough model is 1 (the state would oscillate between C and $A \cup B$ until it reaches $Goal$, which happens almost surely in a finite time period). This estimate is pretty far from the real probability, which equals $P(Goal) + P(A)$, which can be arbitrarily small.

Such pathological processes are not just theoretical problems. We encountered them in the universes we used for testing the algorithm on, so we must deal with them in order to build the net. We also remark that there exist universes such that a realistic Markovian model has precisely the structure of the pathological one we have seen. For example, in a universe in which the agent has to seek a target by alternating between an action to go forward and an action to re-center the target, the oscillations between the states "target centered" and "target not centered" enable one to reach the goal almost surely. Thus, we cannot expect to detect the anomaly just by examining the values of the transition probabilities.

3.4 Construction of the net

3.4.1 Class of the a_i functions

For the planing unit nets we will consider in the rest of this article, each a_i is just a function of a few of the universe parameters, and will be equal to 1 if and only if the vector of parameters has a given value (the units behave as the classifiers proposed in (Booker et al. 1989)). Therefore, one can associate with each a_i a vector $v \in \{\sharp, 0, 1\}^{\mathcal{P}}$ such that :

$$(a_i(x) = 1) \iff (\forall p \in \mathcal{P},\ (x_p = v_p) \text{ or } (v_p = \sharp))$$

3.4.2 Iterative construction of the planing unit net

The construction of the net is done by iteratively increasing the total number of units. It consists of splitting the situation associated with one of the units by replacing this unit by two new units, the values of which depend on one more parameter than the original did. Formally, we will denote by $\mathsf{R}_0, \ldots, \mathsf{R}_k$ the sequence of nets, and $a_1^n, \ldots, a_{n+2}^n$ the activations of the units of R_n. Let us denote by j the index of the unit we are going to split to build R_{n+1} from R_n, and let us denote by p the parameter used for the split (j and p depend on n, we will see later how they are determined). Then we have :

$$\forall x \in \mathcal{S} \left\{ \begin{array}{rcl} a_1^{n+1}(x) & = & a_1^n(x) \\ & \cdots & \\ a_{j-1}^{n+1}(x) & = & a_{j-1}^n(x) \\ a_j^{n+1}(x) & = & a_j^n(x) x_p \\ a_{j+1}^{n+1}(x) & = & a_j^n(x)(1 - x_p) \\ a_{j+2}^{n+1}(x) & = & a_{i-1}^n(x) \\ & \cdots & \\ a_{n+3}^{n+1}(x) & = & a_{n+2}^n(x) \end{array} \right.$$

The unit j of R_n is replaced by two units in R_{n+1}, all the others remaining the same. We will say that R_n is the *father* of R_{n+1}, and that R_{n+1} is a *son* of R_n.

3.4.3 Split selection

When the program is initialized, the net has only two units. The first one is associated with the goal (cf. 3.3.1), the other with its complement. At each step of the construction, one needs to determine which unit to split, and with respect to which parameter. We use a brute-force method consisting of examining all the possible splits, computing a value function for each of them, and keeping the one that maximizes the value.

We try to maximize the probability for the net to reach the goal situation, starting from any initial situation. A natural value function would be the estimation of the probability to reach the goal situation using the policy selected by the net. Considering the Markovian hypothesis, and the stationarity hypothesis, and denoting π the policy chosen by the net R, we obtain :

$$\begin{aligned}\Psi(\mathsf{R}) &= \hat{P}_\pi(\exists t < \infty,\ a_1(t) = 1) \\ &= \textstyle\sum_i \hat{P}_\pi(\exists t < \infty,\ a_1(t) = 1 \mid a_i(0) = 1)\hat{P}(a_i(0) = 1) \\ &= \textstyle\sum_i d_i \hat{P}(a_i(0) = 1)\end{aligned}$$

As we saw in 3.3.3, such an expression using just the Markovian model could provoke dramatic errors by leading to choose a net that over-estimates its probability to reach the goal. To handle this, we propose a value function that compares the qualities of models between the sons and their father.

3.4.4 Differential Efficiency

The Ψ function above estimates the probability to reach the goal, starting from any initial situation and using the optimum policy chosen by the net. One can as well estimate this probability for any other policy just by using a propagation rule similar to (DP) with a fixed π.

Let R be a net, and R' one of its sons. By definition, any policy π of R associates with a unit an action to perform. Hence a policy of R is also a policy of R'. This is clear by considering a policy that associates with every unit common to both R

and R' the same action as π does, and to the two new units of R' the action that π associates with the unit they replace. Then, in view of the preceding discussion, we can estimate with R' the probability to reach the goal using the policy π chosen by R.

Thus, if R' is a son of R, if π is the optimum policy chosen by R, and π' the one chosen by R', we compute with R' the following value function, called *differential efficiency*, to estimate its quality :

$$\Phi(\mathsf{R}', \mathsf{R}) \;=\; \hat{P}_{\pi'}(\exists t < \infty,\ a_1(t) = 1) - \hat{P}_{\pi}(\exists t < \infty,\ a_1(t) = 1) \qquad \text{(DE)}$$

This value function is large if R' estimates a high probability of reaching the goal, but also if it estimates that its father has a low probability of doing so. In particular, if a net estimates its policy is not better than its father's, it has a zero value. Most of the time a large value of this function is related to an error in the Markovian model associated with R and solved by R'. Thus, this split rule takes into account both the quality of the model and the also the efficiency in reaching the goal.

The Kolmogorov-Smirnov test used in U-Tree (McCallum 1996*a*, McCallum 1996*b*) gives large values to splits which substantially change the prevision of reward, and it does not select splits which permit to reach the goal efficiently as DEA does. In most of the tests we have made, U-Tree requires more states than DEA does.

Comparing the cost of these two algorithms, it appears that both consider all possible splits and compute a value which requires the estimation of the d_i in the candidate networks. For the U-Tree algorithm, those d_i are used to estimate the distributions of the rewards (i.e. the distribution of the desire activation of the next activated unit, doing the selected action), whereas DEA uses the d_i to compare the estimates of the probability of reaching the goal. The costs of those two operations are similar and we can, like McCallum, apply a few techniques in order to reduce the number of considered splits ((McCallum 1996*b*), page 93). Finally, the cost of one step is the same for DEA and U-Tree. The latter reduces the total cost by applying few splits at each step. We have not tried this strategy with DEA, and as seen in section 4, in some cases U-Tree performs better if splits are applied sequentially.

4 Results

4.1 Implementation and evaluation protocol

The experiments were done using a program written in C++ with GNU tools on GNU/Linux computers. The kernel simulating the planning unit net is the same whatever the universe it is tested on. On a standard PC computer, for the examples we present here, the maximum computation time required to build the net is less than 15 minutes.

The DEA is tested on four different universes, addressing various categories of problems. For a given universe, during an initial step, the algorithm records the values of the universe's parameters and actions while the agent acts randomly, following reflex rules. The duration of this period, and the random choice of actions depends on the universe[2]. Then, the algorithm builds the net by successively adding units, one at a time. The test protocol consists of estimating the performance of the net after each split. To compare the duration of the training period with the equivalent for dicrete time problems one have to keep in mind the duration of continuous-time actions ; a cycle in a continuous time universe is not equivalent to one action trial.

This repeated evaluation period consists of measuring in 1000 test cases the capability of the agent to reach the goal situation, starting in a random initial situation. The failure is the proportion of attempts for which the agent did not reach the goal situation in less time than a fixed value. This *maximum time before failure* will be indicated for each universe, and is of the order of the minimum time required to reach the goal, starting from the worst situation.

This failure rate is a demanding measure. It quantifies the ability of the net to

[2]The duration of the reflex period is empirically fixed so that the $\lambda_{i,j}^{\alpha}$ coefficients are estimated with enough accuracy.

generate a complete chain of actions from any situation to the goal. Because the choice of action is deterministic, a net which generates all but one action necessary to reach the goal fails. Thus, the net can have an arbitrarily high failure rate even if it determines almost all the necessary actions along the trajectory to the goal.

We compare DEA to U-Tree algorithm developed by A. McCallum (McCallum 1996*b*). The DEA algorithm applies splits one after another, whereas U-Tree simultaneously applies all splits considered useful. Thus we also tested a modified version of U-Tree we call U-Tree * which applies splits sequentially.

4.2 Block game

The block game is a 3×2 grid on which two blocks (X and Y, as shown in figure 4) can move. At each time step, one of the two blocks can move a fraction of the size of a square in four directions (up, right, down, left). Therefore, there are eight actions. The moving blocks cannot penetrate each other, and the upper-left and upper-right squares of the grid constitute obstacles the blocks cannot penetrate neither. The goal situation is reached when block X is in the lower-right square and block Y is in the lower-left square. There are eight parameters, indicating if one of the two blocks covers, partially or totally, one of the four grid squares. Each block can be on two squares, and so can activate two parameters at the same time.

The learning period has a total duration of $250,000$ time steps. At each step, the action performed has probability 0.9 to be the same as the one during the previous step. If it changes, a new action is randomly chosen. Each block can move one eighth of a square size at each time step, so the goal can always be reached in less than 48 time steps. The maximum time before failure has been fixed at 100 steps.

The figure 5 shows the failure rate versus the number of units. A zero failure rate is obtained with a 15 unit net. At this point, the growth process stops because the Φ function is zero on all new nets. The total number of possible configurations of parameter values is 24. Hence, this experiment demonstrates the generalization capabilities of DEA, which is able to solve the task without building one unit for each configuration. The U-Tree (respectively U-Tree *) requires 21 (respectively

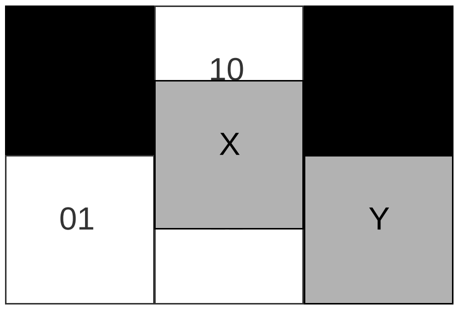


Figure 4: Block game. The blocks can move in the four directions. The upper-left and upper-right squares are obstacles.

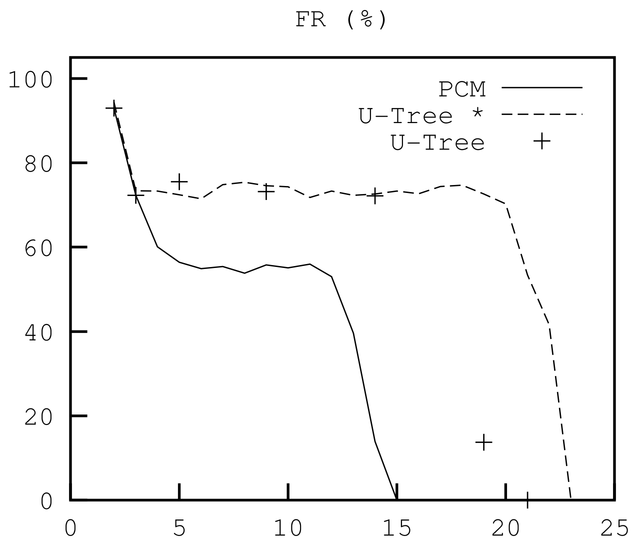


Figure 5: Results for the block game. Failure rate versus the number of units.

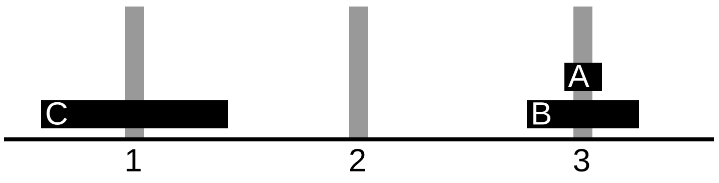


Figure 6: Hanoi towers problem.

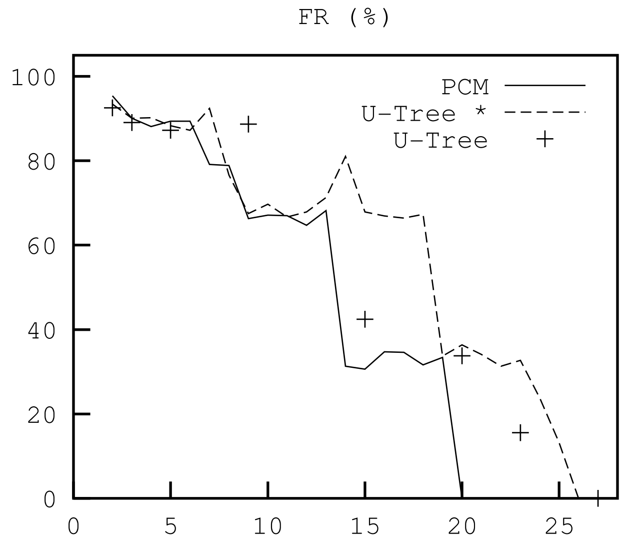


Figure 7: Results for the Hanoi towers problem. Failure rate versus the number of units.

23) units to solve the same task with no error.

4.3 Hanoi towers

The “Hanoi towers” game is a famous goal-planning task. There are three disks of different diameters which can be threaded on three vertical rods (cf. figure 6). The player can move the upper disk of any rod to another rod, but can never put a disk on a smaller one. The goal is to move the three disks to the rod on the right. There are six actions, corresponding to the rod from which the upper disk is removed and the rod it is placed on. An action can have no effect if there is no disk on the first rod chosen, or if it is an illegal move due to the diameter constraint. There are nine parameters, each one associated with a rod and a disk.

The learning period lasts for $30,000$ steps. At each step, an action is chosen randomly. All the actions last one cycle. Because the problem requires in the worst case nine moves, the maximum time before failure has been fixed at 20 time steps.

On the figure 7, the zero failure rate is reached with a 20 unit net. If looking to the average time required to reach the goal, one can observe that it decreases (almost linearly from 5.3 to 4.6 cycles) as the number of units increases whereas the failure rate is already zero. By making the model finer, DEA is able to find shorter paths to the goal. The error rates for both version of U-tree show that this algorithm requires almost an exhaustive model to solve the problem with a null error rate (it builds 26 units and the universe has 27 states).

4.4 Beaver world

The beaver world consists of a square play field containing three trees, two houses and an animat (cf. figure 8). The actions allow the animat to move, to grasp a tree or to release it. The goal situation is reached when the animat is close to a house and a tree at the same time. We define "close to the animat" as closer than a fixed distance corresponding to the circle delimiting the field of view around the animat in the figure.

There are five actions :

- GF makes the animat move forward one hundredth of the play field border length
- T_R and T_L make the animat turn to the right and to the left one hundredth of a complete turn
- Tk makes the animat grasp a tree if there is one close to it
- Pt makes the animat release the tree if it was holding one

The animat perceives the universe with two eyes (its field of view is composed of 5×2 cells, which limits go from $-60°$ to $+60°$ relatively to the direction of the animat) and by touching (it knows if there is a house or a tree close to it and it knows if it is holding a tree). Each object (houses, trees) have a color randomly selected in a set of 8 colors. There are 32 parameters :

- Gl represents the goal
- Hl indicates that the animat is holding a tree
- at indicates that the animat is touching a tree
- $a_{0,0}$ to $a_{4,1}$ indicate that the animat has a tree in one of the 10 cells of its field of view
- bt indicates that the animat is touching a house
- $b_{0,0}$ to $b_{4,1}$ indicate that the animat has a house in one of the 10 cells of its field of view
- c_0 to c_7 indicate which colors are visible in the field of view

The training period lasts for $2,500,000$ steps. During this period, the action performed each step depends on the value of the parameters. If the animat touches a tree (respectively a house) it has a high probability to perform the “grasp tree” action (respectively the “release tree” action). Otherwise, if there is an object in one of its fields of view (tree or house), it has a high probability to go forward ; otherwise it turns left or right with the same probability. This rough strategy enables the animat to explore the square and perform the “grasp tree” and “releases tree” actions in appropriate conditions, frequently enough to estimate statistical parameters. To solve the problem, the animat must first go to a tree and grasp it, then go to a house, and release the tree near the house.

The maximum time required to reach the goal has been empirically measured and is under 200 time steps. The maximum time before failure has been fixed at $1,000$ time steps. The measured total number of parameter value configurations is $62,000$, and DEA builds a net with 8 units which solves the problem with a zero failure rate ; see figure 9.

In order to control its trajectory to a tree or to a house, the animat goes forward when the target is in front of it, and turns in order to put the target back in front as soon as it leaves it. The net built to perform this strategy has two sub-structures dedicated respectively to guidance to a tree and guidance to a house.

Both versions of U-Tree perform very poorly on this task. Comparing the units built by DEA and those built by U-Tree, it appears that whereas the former creates only the required units for the retina, both versions of the latter go on splitting units related to the retina. Since one can improve the model of the universe in term of prediction (but not in term of efficiency to reach the goal) by creating one unit for each possible configuration of the retina's parameters, this behaviour of U-Tree is to be expected.

The reason for such a high failure rate is that, when put in a random state, the beaver has a very high probability to be far from both trees and houses. Thus, a network whose policy is not complete (i.e. makes the animat go to the tree, grasp it and then go to a house and release the tree) has an error rate close to 100%.

4.5 Eye universe

This last universe is an elementary character recognition problem and demonstrates to some extent the classification capabilities of DEA and its relation with active vision (Rimey & Brown 1994, Swain & Stricker 1993). In this universe, the net can move a retina or name the object it has in its field of view. Therefore, we unify goal-planning and classification by adding "answer actions".

The universe consists of a square retina that can move onto a symbol to recognize, which is chosen from a set of four possible characters. This retina is divided into four quarters, each of them holding four geometric feature detectors, which detect, respectively, extremities, angles, curves and T-junctions (cf. figure 10), and can be considered as combinations of elementary border detectors, such as those that exist in the cortex (Hubel & Wiesel 1962). Depending on the character shown to the retina and its location, some of the detectors are activated (cf. figure 11). There are eight different actions, four related to the displacement of the retina in each of the four possible directions (one twenty-th of the retina border length at each step), and four others related to the naming of the character. There are sixteen parameters, each one associated with a quarter of the retina and a detector. The goal is reached when the net correctly gives the name of the character. We have used two different sets of four characters : {1, 2, 3, 4} and {A, B, C, D}.

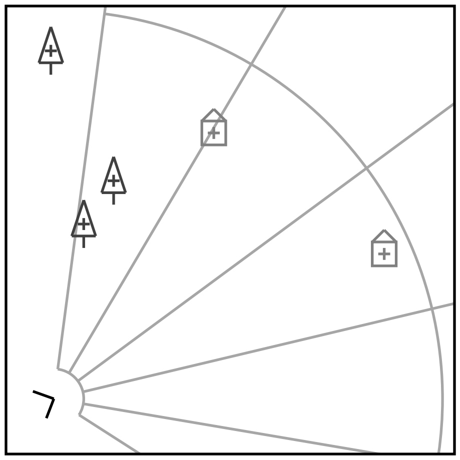


Figure 8: Beaver world. The animat can turn, go forward, grasp and release trees.

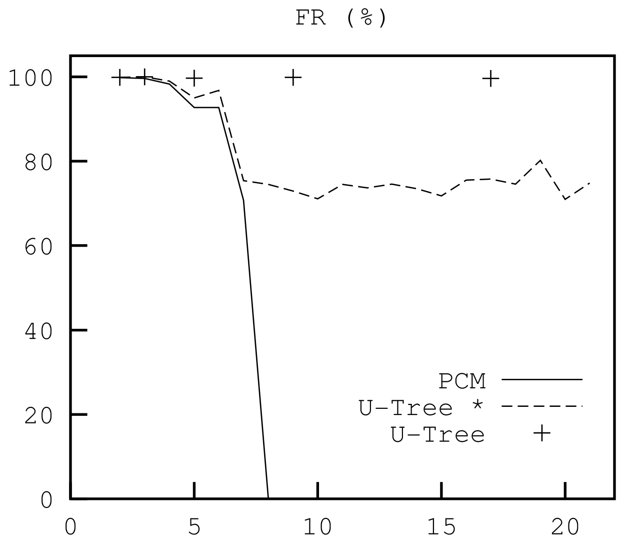


Figure 9: Results for the beaver world. Failure rate versus the number of units.

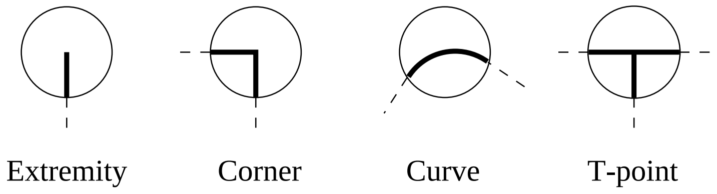


Figure 10: The detectors are associated with four kinds of local geometric configurations.

The characters are piecewise linear, with labels “extremity”, “corner”, “curve” and “t-point” assigned by hand at each vertex. We used the same sets of symbols during the learning and the testing ; therefore we do not prove any form of generalization capability.

The learning period lasts $1,000,000$ time steps , during which the agent alternates between movements and answers. The movements are chosen in such a way that the character is roughly centered : if all the detectors in one half of the retina are not activated, the movements that bring the character in this part of the retina are chosen with a higher probability.

The functioning of the net after the learning period demonstrates the utility of an approach which allows the recognition system to reduce ambiguities by moving the retina on the character. For example, the symbols “2” and “3” activate the same detectors if they are in one quarter of the retina, since both have extremities, corners, and curves, and hence cannot be separated. To solve this problem, the net moves them so that they are on two quarters of the retina (the “2” symbol cannot activate two “corner” detectors, whereas the “3” can). The numbers of possible parameter value configurations are, respectively, 247 and 221 the two sets of characters, and the nets required to solve the problems have 23 and 12 units. On the first set of characters, U-Tree and DEA have similar failure rate, but DEA is more efficient on the second set (cf. figure 12).

4.5.1 Comparison with classification trees

We can compare DEA to algorithms for binary trees by considering a universe like the Eye universe, without actions to modify the data. Let us call Y the class of the object to be recognized. There is one action associated with each class, and the goal is reached if and only if the agent performs the right action. If a is a function of the parameters, put :

$$I(a) \quad = \quad \max_\alpha \ \hat{P}(Y = \alpha \mid a(0) = 1) \ \hat{P}(a(0) = 1)$$

It’s easy to show that, denoting by a the activation of the column to split, and by a' and a'' the activations of the two created columns, we obtain :

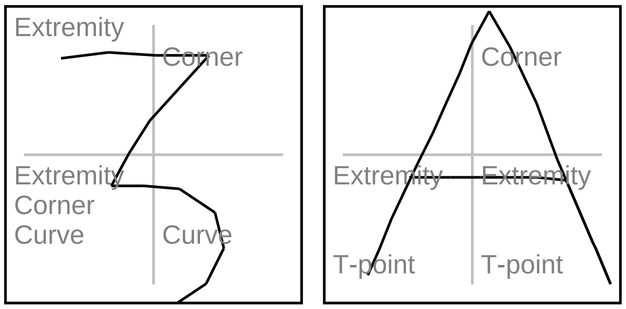
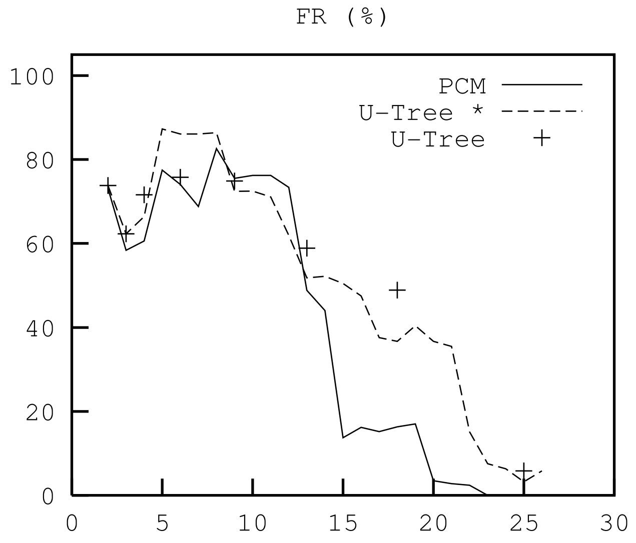


Figure 11: Two examples of characters in the eye universe. Each quarter of the retina possesses four different features detectors.



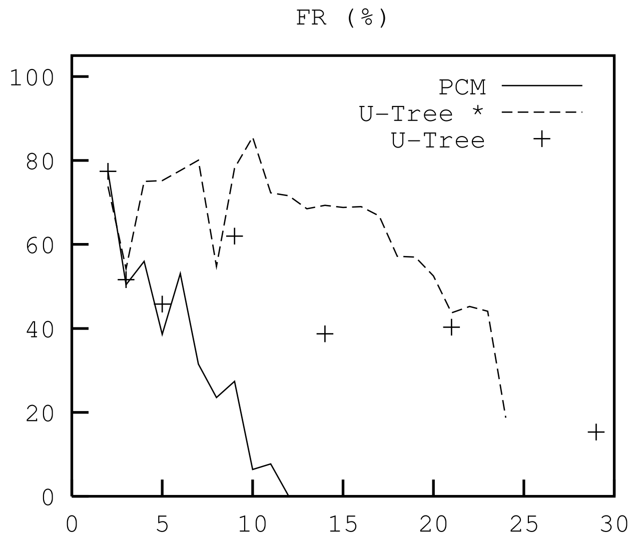


Figure 12: Results for the eye universe for the sets of characters {1, 2, 3, 4} (top) and {A, B, C, D} (bottom). Failure rate versus number of units.

$$\Phi(\mathsf{R}',\mathsf{R}) \quad = \quad I(a') + I(a'') - I(a)$$

This corresponds to a "splitting rule" that tries to minimize the error rate at each step. This rule has been described and criticized in (Breiman, Friedman, Olshen & Stone 1984). The main defect is that reducing the misclassification rate at each step does not produce a good *global* strategy. For example, two splits which lead to the same error rate are equivalent for this splitting rule even if one of the splits produces a pure unit with respect to Y, which is a better configuration since such a unit is optimal and ignored afterwards. The standard method used to solve this problem consists of using criteria such as Shannon entropy which yield better estimates of the "purity" of units because it depends on all the class probabilities and not only on the probability of the more probable state.

We believe that a good criterion for the construction of the net should reduce to the Shannon entropy-based rule or an equivalent one in this particular universe.

5 Discussion

Starting from analogies with cortical structures, we have developped a planning network with a formalization on the MDP theory. We have also proposed a procedure to construct the net by adding units, equivalent to splitting iteratively the perception state. We have shown that this algorithm can solve both goal-planning and classification problems, demonstrating that those functions can be combined into one structure.

The comparison with U-Tree, which addresses the same tasks, underscores the interest of DEA, which attempts to maximize both the quality of the Markovian model and the probability of reaching the goal. With such a strategy, DEA is able to generate more compact models (with a smaller number of states), and needs a smaller number of transition probabilitie estimations to create a fully efficient Markovian model dedicated to one goal. In some critical cases (cf. beaver example) increasing the quality of the Markovian model, without taking into account the efficiency relative to the goal, leads to an excessive and useless creation of states.

This algorithm is a simple example of a large class of algorithms using the general idea of efficiency maximization. It could be improved by adding a rule for unit fusion allowing to back-track the splitting process, and cancel a split when it appears useless.

6 Thanks

We want to thank Yves Burnod for his advice and participation, and Donald Geman for his help during the preparation of this paper.

References

Barto, A. G., Sutton, R. S. & Watkins, C. J. C. H. (1989), Learning and sequential decision making, Technical Report 89-95, COINS.

Bellman, R. (1957), *Dynamic Programming*, Princeton University Press, Princeton, NJ.

Bertsekas, D. P. & Tsitsiklis, J. N. (1989), *Parallel and Distributed Computation*, Prenctice-Hall International Edition.

Booker, L. B., Goldberg, D. E. & Holland, J. H. (1989), 'Classifier systems and genetic algorithms', *Artificial Intelligence* **40**, 235–282.

Breiman, L., Friedman, J., Olshen, R. & Stone, C. (1984), *Classification And Regression Trees*, Wadsworth, Statistics/probability series.

Burnod, Y. (1989), *An Adaptative Neural Network : the Cerebral Cortex*, Masson collection biologie théorique.

Chapman, D. & Kaelbling, L. P. (1991), Input generalization in delayed reinforcement learning : An algorithm and performance comparisons, *in* 'Proceedings of the International Joint Conference On Artificial Intelligence, Sydney, Australia'.

Dehaene, S. & Changeux, J.-P. (1997), ‘A hierarchical neuronal network for planning behavior’, *Proc. Natl. Acad. Sci. USA* **94**, 13293–13298.

Guigon, E., Grandguillaume, P., Otto, I., Boutkhil, L. & Burnod, Y. (1994), ‘Neural network models of cortical functions based on the computational properties of the cerebral cortex’, *Journal of Physiology, Paris* **88**, 291–308.

Howard, R. A. (1960), *Dynamic Programming and Markov Processes*, MIT Press, Cambridge.

Hubel, D. H. & Wiesel, T. N. (1962), ‘Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex’, *Journal of Physiology* **160**, 106–154.

Kaelbling, L. P. (1993), *Learning in Embedded Systems*, MIT Press, Cambridge.

Kaelbling, L. P., Littman, M. L. & Moore, A. W. (1996), ‘Reinforcement learning : A survey’, *Journal of Artificial Intelligence Research* **4**, 237–285.

Koenig, S. & Simmons, R. G. (1996), ‘The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms’, *Machine Learning* **22**, 227–250.

Laird, J. E., Newell, A. & Rosenbloom, P. S. (1987), ‘Soar : An architecture for general intelligence’, *Artificial Intelligence* **33**, 1–64.

McCallum, A. K. (1996*a*), Learning to use selective attention and short-term memory in sequential tasks, *in* ‘From animals to animats : Proceedings of the Fourth International Conference on Simulation of Adaptative Behavior’, MIT Press.

McCallum, A. K. (1996*b*), Reinforcement Learning with Selective Perception and Hidden State, PhD thesis, University of Rochester, Rochester, New York.

Moore, A. W. & Atkeson, C. G. (1993), ‘Prioritized sweeping : Reinforcement learning with less data and less time’, *Machine Learning* **13**, 103–130.

Moore, A. W. & Atkeson, C. G. (1995), 'The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces', *Machine Learning* **21**.

Mozer, M. C. & Bachrach, J. (1991), 'Slug : A connectionnist architecture for inferring the structure of finite-state environments', *Machine Learning* **7**, 139–160.

Piaget, J. (1964), *Six Etudes de Psychologie*, Editions Denoel.

Putterman, M. L. (1994), *Markov Decision Processes : Discrete Stochastic Dynamic Programming*, Wiley, New York.

Rimey, R. R. & Brown, C. M. (1994), 'Control of selective perception using bayes nets and decision theory', *International Journal of Computer Vision* **12:2/3**, 173–207.

Ripley, B. D. (1994), 'Neural networks and related mehods for classification', *Journal of the Royal Society B* **56(3)**, 409–456.

Sutton, R. S. (1990), Integrated architecture for learning, planning, and reacting based on approximating dynamic programming, *in* 'Proceedings of the Seventh Int. Conf. on Machine Learning', Morgan Kaufmann, pp. 216–224.

Swain, M. J. & Stricker, M. A. (1993), 'Promising directions in active vision', *International Journal of Computer Vision* **11:2**, 109–126.