# Recognizing internal states of other agents to anticipate and coordinate interactions

Filipo Studzinski Perotto

SUPINFO International University, Toulouse, France
filipo.perotto@gmail.com

**Abstract.** In multi-agent systems, anticipating the behavior of other agents constitutes a difficult problem. In this paper we present the case where a cognitive agent is inserted into an unknown environment composed of different kinds of other objects and agents; our cognitive agent needs to incrementally learn a model of the environment dynamics, doing it only from its interaction experience; the learned model can be then used to define a policy of actions. It is relatively easy to do so when the agent interacts with static objects, with simple mobile objects, or with trivial reactive agents; however, when the agent deals with other complex agents that may change its behaviors according to some non directly observable internal states (like emotional or intentional states), the construction of a model becomes significantly harder. The complete system can be described as a *Factored and Partially Observable Markov Decision Process* (FPOMDP); our agent implements the *Constructivist Anticipatory Learning Mechanism* (CALM) algorithm, and the experiment (called *meph*) shows that the induction of non-observable variables enable the agent to learn a deterministic model of most of the universe events, allowing it to anticipate other agents actions and to adapt to them, even if some interactions appear as non-deterministic in a first sight.

**Keywords.** Factored Partially Observable Markov Decision Process (FPOMDP), Constructivist Learning Mechanisms, Anticipatory Learning, Model-Based RL.

## 1 Introduction

Trying to escape from AI classic (and simple) maze problems toward more sophisticated (and therefore more complex and realistic) agent-based universes, we are led to consider some complicating conditions: (a) the situatedness of the agent, which is immersed into an unknown universe, interacting with it through limited sensors and effectors, without any holistic perspective of the complete environment state, and (b) without any *a priori* model of the world dynamics, which forces it to incrementally discover the effect of its actions on the system in an on-line experimental way; to make matters worse, the universe where the agent is immersed can be populated by different kinds of objects and entities, including (c) other complex agents, and in this case, the task of learning a predictive model becomes considerably harder.

We are especially concerned with the problem of discovering the existence of other agents' internal variables, which can be very useful to understand their behavior. Our cognitive agent needs to incrementally learn a model of its environment dynamics, and the interaction with other agents represents an important part of it. It is relatively easy to construct a model when the agent interacts with static objects, with simple mobile objects, or with trivial reactive agents; however, when dealing with other complex agents which may change its behaviors according to some non directly observable internal properties (like emotional or intentional states), the construction of a model becomes significantly harder. The difficulty increases because the reaction of each agent can appears to our agent as a non deterministic behavior, regarding the information provided by the perceptive elements of the situation.

We can anticipate at least two points of interest addressed by this paper: the first one is about concept creation, the second one is about agent inter-subjectivity. In the philosophical and psychological research community concerned with cognitive issues, the challenge of understanding the capability to develop new abstract concepts have always been a central point in most theories about how the human being can deal and adapt itself to a so complex and dynamical environment as the real world (Murphy, 2002), (Piaget, 1947). In contrast to the kind of approach usually adopted in AI, which easily slip into the strategy of treating exceptions and lack of information by using probabilistic methods, many cognitive scientists insist that the human mind strategy looks more like accommodating the disturbing events observed in the reality by improving his/her model with new levels of abstraction, new representation elements, and new concepts. Moreover, the intricate problem of dealing with other complex agents has also been studied by cognitive science for some time, from psychology to neuroscience. A classical approach to it is the famous “ToM” assumption (Astington, et al. 1988), (Bateson, 1972), (Dennett, 1987)

which claims that the human being have developed the capability to attribute mental states to the others, in order to represent their beliefs, desires and intentions, and so being able to understand their behavior.

In this paper, we use the *Constructivist Anticipatory Learning Mechanism* (CALM), defined in (Perotto, 2010), to solve the "*meph* problem", where a cognitive agent is inserted into an environment constituted of other objects and also of some other agents, that are non-cognitive in the sense that they do not learn anything, but that are similar to our agent in terms of structure and possible behaviors. CALM is able to build a descriptive model of the system where the agent is immersed, inducting, from the experience, the structure of a factored and partially observable Markov decision process (FPOMDP). Some positive results have been achieved due to the use of 4 integrated strategies (Perotto, 2010), (Perotto et al. 2007), (Perotto; Alvares, 2007): (a) the mechanism takes advantage of the situated condition presented by the agent, constructing a description of the system regularities relatively to its own point of view, which allows to set a good behavior policy without the necessity of "mapping" the entire environment; (b) the learning process is anchored on the construction of an anticipatory model of the world, what could be more efficient and more powerful than traditional "model free" reinforcement learning methods, that directly learn a policy; (c) the mechanism uses some heuristics designed to *well structured* universes, where conditional dependencies between variables exist in a limited scale, and where most of the phenomena can be described in a deterministic way, even if the system as a whole is not, representing what we call a partially deterministic environment; this characteristic seems to be widely common in real world problems; (d) the mechanism is prepared to discover the existence of hidden or non-observable properties of the universe, which cannot be directly perceived by the agent sensors, but that can explain some observed phenomena. This last characteristic is fundamental to solve the problem presented in this article because it enables our agent to discover the existence of internal states in other agents, which is necessary to understand their behavior and then to anticipate it. Further discussion about situatedness can be found in (Wilson; Clark, 2008), (Beer, 1995), and (Suchman, 1987).

Thus, the basic idea concerning this paper is to describe the algorithm CALM, proposed in (Perotto, 2010), presenting its features, and placing it into the *Markov Decision Process* (MDP) framework panorama. The discussion is supported, on one side, by these introductory philosophical conjectures, and on the other side, by the *meph* experiment, which creates a multi-agent scenario, where our agent needs to induce the existence of internal variables to the other agents. In this way, the paper presents some positive results in both theoretical and practical aspects. Following the paper, section 2 overviews the MDP framework, section 3 describes the CALM learning mechanism, section 4 introduces the experiment and shows the acquired results, and section 5 concludes the paper, arguing that the discover and induction of hidden properties of the system can be a promising strategy to model other agents internal states.

## 2 Markov Decision Process Framework

*Markov Decision Process* (MDP) and its extensions constitute a quite popular framework, largely used for modeling *decision-making* and *planning* problems. An MDP is typically represented as a discrete stochastic state machine; at each time cycle the machine is in some state $s$; the agent interacts with the process by choosing some action $a$ to carry out; then, the machine changes into a new state $s'$, and gives the agent a corresponding reward $r$; a given transition function $\delta$ defines the way the machine changes according to $s$ and $a$. Solving an MDP is finding the optimal (or near-optimal) policy of actions in order to maximize the rewards received by the agent over time. When the MDP parameters are completely known, including the reward and the transition functions, it can be mathematically solved by *dynamic programming* (DP) methods. When these functions are unknown, the MDP can be solved by *reinforcement learning* (RL) methods, designed to learn a policy of actions on-line, i.e. at the same time the agent interacts with the system, by incrementally estimating the utility of state-actions pairs and then by mapping situations to actions (Sutton, Barto 1998).

### 2.1 The Classic MDP

*Markov Decision Process* first appeared (in the form we know) in the late 1950s (Bellman, 1957), (Howard, 1960), reaching a concrete popularity in the *Artificial Intelligence* (AI) research community from the 1990s (Puterman, 1994). Currently the MDP framework is widely used in the domains of *Automated Control*, *Decision-Theoretic Planning* (Blythe, 1999), and *Reinforcement Learning* (Feinberg, Shwartz, 2002). A "standard MDP" represents a system through the discretization and enumeration of its state space, similar to a *state machine* in which the transition function can be non-deterministic. The flow of an MDP (the transition between states) depends only on the system current state and on the action taken

by the agent at the time. After acting, the agent receives a reward signal, which can be positive or negative if certain particular transitions occur.

However, for a wide range of complex (including real world) problems, the complete information about the exact state of the environment is not available. This kind of problem is often represented as a *Partially Observable Markov Decision Process* (POMDP) (Kaelbling et al., 1998). The idea of representing non-observable elements in a MDP is not new (Astrom, 1965), (Smallwood; Sondik, 1973), but became popular with the revived interest on the framework, occurred in the 1990s (Christman, 1992), (Kaelbling et al., 1994, 1998). The POMDP provides an elegant mathematical framework for modeling complex decision and planning problems in stochastic domains in which the system states are observable only indirectly, via a set of imperfect, incomplete or noisy perceptions. In a POMDP, the set of observations is different from the set of states, but related to them by an observation function, i.e. the underlying system state *s* cannot be directly perceived by the agent, which has access only to an observation *o*. The POMDP is more powerful than the MDP in terms of modeling (i.e. a larger set of problems can be described by a POMDP than by an MDP), but the methods for solving them are computationally even more expensive, and thus applicable in practice only to very simple problems (Hauskrecht, 2000), (Meuleau et al., 1999), (Shani et al., 2005).

The main bottleneck about the use of MDPs or POMDPs is that representing complex problems implies that the state space grows-up and quickly becomes intractable. Real-world problems are generally complex, but fortunately, most of them are quite well-structured. Many large MDPs have significant internal structure, and can be modeled compactly if the structure is exploited in the representation. The factorization of states is an approach to exploit this characteristic. In the factored representation, a state is implicitly described by an assignment to some set of state variables. Thus, the complete state space enumeration is avoided, and the system can be described referring directly to its properties. The factorization of states enable to represent the system in a very compact way, even if the corresponding MDP is exponentially large (Guestrin et al. 2003), (Shani et al. 2008). When the structure of the *Factored Markov Decision Process* (FMDP) (Boutilier et al. 2000) is completely described, some known algorithms can be applied to find good policies in a quite efficient way (Guestrin et al., 2003). However, the research concerning the discover of the structure of an underlying system from incomplete observation is still incipient (Degris et al., 2006), (Degris, Sigaud, 2010).

### 2.2 Factored and Partially Observable MDP

In order to increase the range of representable problems, the classic MDP model can be extended to include factorization of states and partial observation, and it can be so called a *Factored Partially Observable Markov Decision Process* (FPOMDP). In order to be factored, the description of a given state *s* in the original model will be decomposed and replaced by a set $\{x_1, x_2, \ldots x_n\}$ in the extended model; the action a becomes a set $\{c_1, c_2, \ldots c_m\}$; the reward signal *r* becomes $\{r_1, r_2, \ldots r_k\}$; and the transition function $\delta$ is replaced by a set of transformation functions $\{T_1, T_2, \ldots T_n\}$.

A FPOMDP (Degris; Sigaud, 2010) can be formally defined as a 4-tuple $\{X, C, R, T\}$. The finite non-empty set of system properties or variables $X = \{X_1, X_2, \ldots X_n\}$ is divided into two subsets, $X = P \cup H$, where the subset *P* represents the observable properties (those that can be accessed through the agent sensory perception), and the subset *H* represents the hidden or non-observable properties; each property $X_i$ is associated to a specified domain, which defines the values the property can assume. $C = \{C_1, C_2, \ldots C_m\}$ represents the controllable variables, composing the agent actions, $R = \{R_1, R_2, \ldots R_k\}$ is the set of (factored) reward functions, in the form $R_i : P_i \rightarrow \mathbb{R}$, and $T = \{T_1, T_2, \ldots T_n\}$ is the set of transformation functions, as $T_i : X \times C \rightarrow X_i$ , defining the system dynamics. Each transformation function can be represented as a *Dynamic Bayesien Network* (DBN) (Dean; Kanazawa, 1989), which is an acyclic, oriented, two-layers graph. The first layer nodes represent the environment state in time *t*, and the second layer nodes represent the next state, in *t+1* (Boutilier et al. 2000). A stationary policy $\pi$ is a mapping $X \rightarrow C$ where $\pi(x)$ defines the action to be taken in a given situation. The agent must learn a policy that optimizes the cumulative rewards received over a potentially infinite time horizon. Typically, the solution $\pi^*$ is the policy that maximizes the expected discounted reward sum, as indicated in the classical Bellman optimality equation (1957), here adapted to our FPOMDP notation.

$$V^{\pi^*}(x) = R(x) + max_c [\, \gamma \, . \, \Sigma_{x'} P(x' \mid x, c) \, . \, V^{\pi^*}(x') \,]$$

In this paper, we consider the case where the agent does not have an *a priori* model of the universe where it is situated (i.e. it does not have any idea about the transformation function), and this condition forces it to be endowed with some capacity of learning, in order to be able to adapt itself to the system. Even if there is a large research community studying model-free methods (that directly learn a policy of actions), in this work we adopt a model-based method, through which the agent must learn a descriptive

and predictive model of the world, and so define a behavior strategy based on it. Learning a predictive model is often referred as learning the structure of the problem, which is an important research objective into the MDP framework community (Degris et al. 2006), as well as in related approaches like *Induction of Decision Trees* or *Decision Graphs* (Jensen; Graven-Nielsen, 2007), *Bayesian Networks* (BN) (Pearl, 2000), (Friedman; Koller, 2003) and *Influence Diagrams* (Howard; Matheson, 1981).

In this way, when the agent is immersed in a system represented as a FPOMDP, the complete task for its anticipatory learning mechanism is both to create a model of the transformation function, and to define an optimal (or sufficiently good) policy of actions. The transformation function can be described by a *dynamic bayesian network*, i.e. an acyclic, oriented, two-layers graph, where the first layer nodes represent the environment situation in time *t*, and the second layer nodes represent the next situation, in time *t+1*. A policy $\pi : X \rightarrow C$ defines the behavior to be taken in each given situation (the policy of actions). Several algorithms create stochastic policies, and in this case the action to take is defined by a probability. Degris and Sigaud (2010) present a good overview of the use of this representation in artificial intelligence, referring several related algorithms designed to learn and solve factored FMDPs and FPOMDPs, including both the algorithms designed to calculate the policy given the model (Boutilier et al. 2000), (Boutilier; Poole, 1996), (Hansen; Feng, 2000), (Poupart; Boutilier, 2004), (Hoey et al., 1999), (St-Aubin et al. 2000), (Guestrin et al. 2001), (Sim et al. 2008), and (Shani et al., 2008) and the algorithms designed to discover the structure of the system (Degris et al., 2006), (Degris; Sigaud, 2010), (Strehl et al., 2007), and (Jonsson; Barto, 2005).

## 3 Constructivist Anticipatory Learning Mechanism

The constructivist anticipatory learning mechanism (CALM), detailedly described in (Perotto, 2010), is a mechanism developed to enable an agent to learn the structure of an unknown environment where it is situated, trough observation and experimentation, creating an *anticipatory model of the world,* which will be represented as an FPOMDP. CALM operates the learning process in an *active* and *incremental* way, where the agent needs to choose between alternative actions, and learn the world model as well as the policy at the same time it actuates. There is no separated previous training time; the agent has an unique uninterrupted interactive experience into the system, quite similarly to real life problems. In other words, it must performing and learning at the same time.

The problem can be divided into two tasks: first, building a world model, i.e. to induce a structure which represents the dynamics of the system (composed by agent-environment interactions). Second, to establish a behavioral policy, i.e. to define the actions to do at each possible different state of the system, in order to increase the estimated rewards received over time.

The task becomes harder because the environment is only partially observable, from the point of view of the agent, constituting an FPOMDP. In this case, the agent has perceptive information from a subset of sensory variables, but the system dynamics depends also on another subset of hidden variables. To be able to create the world model, the agent needs, beyond discover the regularities of the phenomena, also discover the existence of non-observable variables that are important to understand the system evolution. In other words, learning a model of the world is more than describing the environment dynamics (the rules that can explain and anticipate the observed transformations), it is also discovering the existence of hidden properties (once they influence the evolution of the observable ones), and finally find a way to deduce the values of these hidden properties.

If the agent can successfully discover and describe the hidden properties of the FPOMDP which it is dealing with, then the world becomes treatable as a FMDP, and there are some known algorithms able to efficiently calculate the optimal (or near-optimal) policy. The algorithm to calculate the policy of actions used by CALM is similar to the one presented by (Degris, Sigaud, 2006). On the other hand, the main challenge is to discover the structure of the problem based on the on-line observation, and CALM do it using representations and strategies inspired on (Drescher, 1993).

### 3.1 Knowledge Representation

CALM tries to reconstruct, by experience, each system transformation function $T_i$, representing it by an *anticipation tree*, which in turn is composed by *schemas*. Each *schema* represent some perceived regularity occurring in the environment, i.e. some regular event checked by the agent during its interaction with the world. A schema is composed by three vectors: $\Xi$ = (*context* $\wedge$ *action* $\rightarrow$ *expectation*). The context vector has each of their elements linked with a sensor. The action vector is linked with the effectors. The expectation represents the value expected for some specific sensor in the next time. In a specific schema, the *context* vector represents the set of equivalent situations where the schema is

applicable. The *action* vector represents a set of similar actions that the agent can carry out in the environment. The *expectation* vector represents the expected result after executing the given action in the given context. Each element vector can assume any value in a discrete interval defined by the respective sensor or effector. In addition, the context vector can incorporate some "synthetic elements" not linked to any sensor but representing abstract or non-sensory properties which the existence is induced by the mechanism.

Some elements in these vectors can undertake an "undefined value". For example, an element linked with a binary sensor must have one of three values: *true*, *false* or *undefined* (represented, respectively, by '1', '0' and '#'). In both the context and action vectors, '#' represents something ignored, not relevant to make the anticipations. There is compatibility between a schema and a certain situation when the schema's context vector has all defined elements equal to those of the agent's perception.

In the expectation vector, '#' means that the element is not deterministically predictable. The undefined value generalizes the schema because it allows to ignore some properties to represent a set of situations. Another symbols can be used to represent some special situations, in a way to reduce the number of schemas; it is the case of the symbol '=', used to indicate that the value of the expected element does not change in the specified context.

The use of undefined values makes possible the construction of an *anticipation tree*. Each node in that tree is a schema, and relations of generalization and specialization guide its topology (quite similar to decision trees or discrimination trees). The root node represents the most generalized situation, which has the context and action vectors completely undefined. Adding one level in the tree is to specialize one generalized element, creating a branch where the undefined value is replaced by the different possible defined values. This specialization occurs either in the context vector or in the action vector. In this way, CALM divides the state space according to the different expectations of changing, grouping contexts and actions with its respective transformations. The tree evolves during the agent's life, and it is used by the agent, even if until under construction, to take its decisions, and in consequence, to define its behavior. The structure of the schemas and an example of their organization as an anticipation tree are presented in Figure 1.

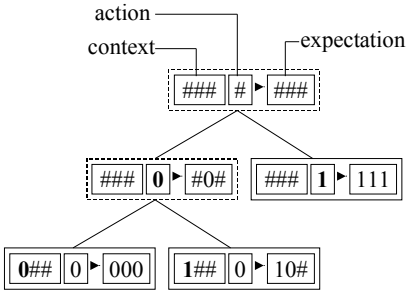


Figure 1: the anticipation tree; each node is a schema composed of three vectors: context, action and expectation; the leaf nodes are decider schemas.

The context in which the agent is at a given moment (perceived through its sensors) is applied in the tree, exciting all the schemas that have a compatible context vector. This process defines a set of excited schemas, each one suggesting a different action to do in the given situation. CALM will choose one to activate, performing the defined action through the agent's effectors. The algorithm always chooses the compatible schema that has the most specific context, called decider schema, which is the leaf of a differentiated branch. This decision is taken based on the calculated utility of each possible choice. There are two kinds of utility: the first one estimates the discounted sum of rewards in the future following the policy, the second one measures the exploration benefits. The utility value used to take the decision depends on the circumstantial agent strategy (exploiting or exploring). The mechanism has also a kind of generalized episodic memory, which represents (in a compact form) the specific and real situations experimented in the past, preserving the necessary information to correctly constructs the tree. The implementation of a feasible generalized episodic memory is not evident; it can be very expensive to remember episodes. However, with some strong but well chosen restrictions (like limiting dependency representation) it can be computationally viable.

### 3.2 Anticipation Tree Construction Methods

The learning process happens through the refinement of the set of schemas. The agent becomes more adapted to its environment as a consequence of that. After each experienced situation, CALM checks if the result (context perceived at the instant following the action) is in conformity to the

expectation of the activated schema. If the anticipation fails, the error between the result and the expectation serves as parameter to correct the model. In the schematic tree topology, the context and action vectors are taken together. This concatenated vector identifies the node in the tree, which grows up using a top-down strategy. The context and action vectors are gradually specialized by differentiation, adding, each time, a new relevant feature to identify the category of the situation. In general there is a shorter way starting with an empty vector and searching for the probably few relevant features than starting with a full vector and having to waste energy eliminating a lot of useless elements. Selecting the good set of relevant features to represent some given concept is a well known problem in AI, and the solution is not easy, even by approximated approaches. To do it, CALM adopts a forward greedy selection (Blum and Langley 1997), using the data registered in the generalized episodic memory.

The expectation vector can be seen as a label in each decider schema, and it represents the predicted anticipation when the decider is activated. The evolution of expectations in the tree uses a bottom-up strategy. Initially all different expectations are considered as different classes, and they are gradually generalized and integrated with others. The agent has two alternatives when the expectation fails. In a way to make the knowledge compatible with the experience, the first alternative is to try to divide the scope of the schema, creating new schemas, with more specialized contexts. Sometimes it is not possible and then it reduces the schema expectation.

Three basic methods compose the CALM learning function, namely: *differentiation*, *adjustment*, and *integration*. Differentiation is a necessary mechanism because a schema responsible for a context too general can hardly make precise anticipations. If a general schema does not work well, the mechanism divides it into new schemas, differentiating them by some element of the context or action vector. In fact, the differentiation method takes an unstable decider schema and changes it into a two level sub-tree. The parent schema in this sub-tree preserves the context of the original schema. The children, which are the new decider schemas, have their context vectors a little bit more specialized than their parent. They attribute a value to some undefined element, dividing the scope of the original schema. Each one of these new deciders engages itself in a part of the domain. In this way, the previous correct knowledge remains preserved, distributed in the new schemas, and the discordant situation is isolated and treated only in its specific context. Differentiation is the method responsible to make the anticipation tree grows up. Each level of the tree represents the introduction of some new constraint. The algorithm needs to choose what will be the differentiator element, and it could be from either the context vector or the action vector. This differentiator needs to separate the situation responsible for the disequilibrium from the others, and the algorithm chooses it by calculating the information gain, and considering a limited (parametrized) range of interdependencies between variables. Figure 2 illustrates the differentiation process.

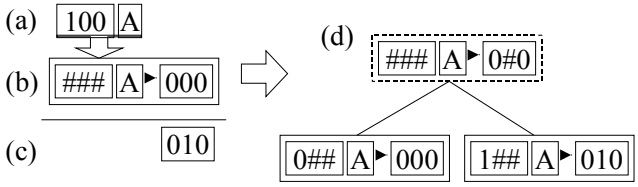


Figure 2. Differentiation method; (a) experimented situation and action; (b) activated schema; (c) real observed result; (d) sub-tree generated by differentiation.

When some schema fails and it is not possible to differentiate it in any way, then CALM executes the adjustment method. This method reduces the expectations of an unstable decider schema in order to make it reliable again. The algorithm simply compares the activated schema's expectation and the real result perceived by the agent after the application of the schema, setting the incompatible expectation elements to the undefined value ('#'). The adjustment method changes the schema expectation (and consequently the anticipation predicted by the schema). Successive adjustments can reveal some unnecessary differentiations. Figure 3 illustrates that.

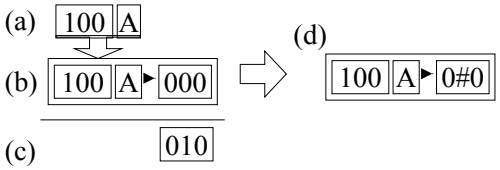


Figure 3. Adjust method; (a) experimented situation and action; (b) activated schema; (c) real observed result; (d) schema expectation reduction after adjust.

In this way, the schema expectation can change (and consequently the class of the situation represented by the schema), and the tree maintenance mechanism needs to be able to reorganize the tree

when this change occurs. Therefore, successive adjustments in the expectations of various schemas can reveal unnecessary differentiations. When CALM finds a group of schemas with similar expectations to approach different contexts, the integration method comes into action, trying to join these schemas by searching for some unnecessary common differentiator element, and eliminating it. The method operates as shown in figure 4.

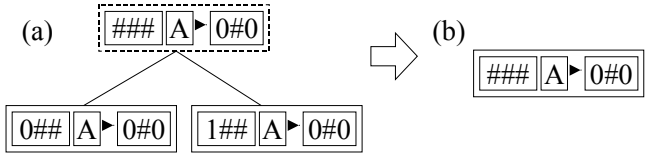


Figure 4. Integration method; (a) sub-tree after some adjust; (b) an integrated schema substitutes the sub-tree.

### 3.3 Dealing with the Unobservable

When CALM reduces the expectation of a given schema by adjustment, it supposes that there is no deterministic regularity following the represented situation in relation to these incoherent elements, and the related transformation is unpredictable. However, sometimes a prediction error could be explained by considering the existence of some abstract or hidden property in the environment, which could be useful to differentiate an ambiguous situation, but which is not directly perceived by the agent sensors. So, before adjusting, CALM supposes the existence of a non-sensory property in the environment, which will be represented as a synthetic element. When a new synthetic element is created, it is included as a new term in the context and expectation vectors of the schemas. Synthetic elements suppose the existence of something beyond the sensory perception, which can be useful to explain non-equilibrated situations. They have the function of amplifying the differentiation possibilities.

In this way, when dealing with partially observable environments, CALM has two additional challenges: (a) inferring the existence of unobservable properties, which it will represent by synthetic elements, and (b) including these new elements into its predictive model. A good strategy to do this task is looking at the historical information. In the case where the POMDP is completely deterministic, it is possible to find sufficient little pieces of history to distinguish and identify all the underlying states (Holmes; Isbell, 2006), and we suppose that it is similar when the POMDP in non-deterministic but well structured.

CALM introduces a method called *abstract differentiation*. When a schema fails in its prediction, and when it is not possible to differentiate it by the current set of considered properties, then a new boolean synthetic element is created, enlarging the context and expectation vectors. Immediately, this element is used to differentiate the incoherent situation from the others. The method attributes arbitrary values to this element in each differentiated schema. These values represent the presence or absence of some non-observable condition, necessary to determine the correct prediction in the given situation. The method is illustrated in figure 5, where the new elements are represented by card suites.

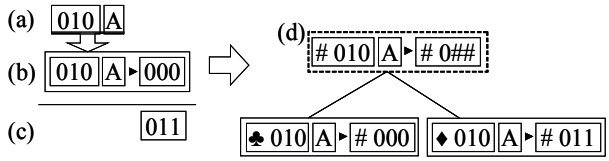


Figure 5. Synthetic element creation method; (d) incremented context and expectation vectors, and differentiation using synthetic element.

Once a synthetic element is created, it can be used in next differentiations. A new synthetic element will be created only if the existing ones are already saturated. To avoid the problem of creating infinite new synthetic elements, CALM can do it only until a determined limit, after which it considers that the problematic anticipation is not deterministically predictable, undefining the expectation in the related schemas by adjustment. Figure 6 explains didactically the idea behind synthetic element creation.

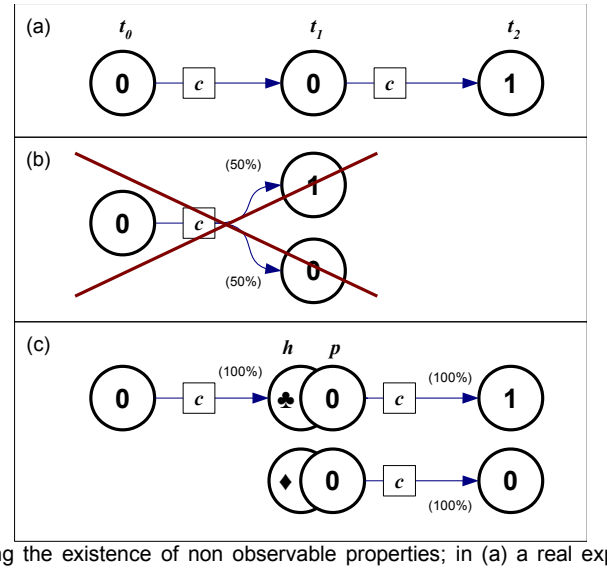


Figure 6: discovering the existence of non observable properties; in (a) a real experienced sequence; in (b) what CALM does not do (the attribution of a probability); in (c) the creation of a synthetic element in order to explain the observed difference.

The synthetic element is not associated to any sensory perception. Consequently, its value cannot be observed. This fact can place the agent in ambiguous situations, where it does not know whether some relevant but not observable condition (represented by this element) is present or absent. Initially, the value of a synthetic element is verified *a posteriori* (i.e. after the execution of the action in an ambiguous situation). Once the action is executed and the following result is verified, then the agent can rewind and deduce what was the situation really faced in the past instant (disambiguated). Discovering the value of a synthetic element after the circumstance where this information was needed can seem useless, but in fact, this delayed deduction gives information to another method called *abstract anticipation*. If the non-observable property represented by this synthetic element has a regular dynamics, then the mechanism can propagate the deduced value back to the schema activated in the immediate previous instant. The deduced synthetic element value will be included as a new anticipation in the previous activated schema. Figure 7 shows how this new element can be included in the predictive model.

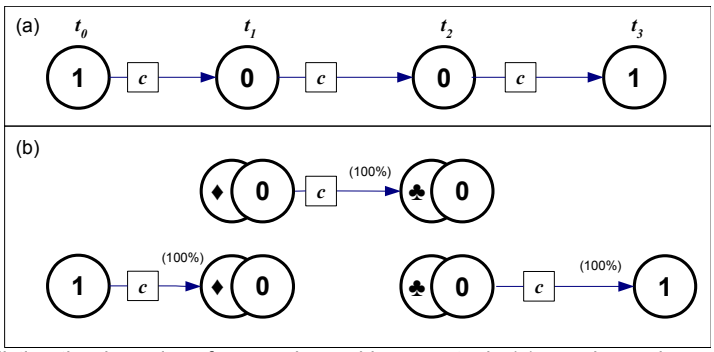


Figure 7: predicting the dynamics of a non observable property; in (a) a real experienced sequence; in (b) the pieces of knowledge that can explain the logic behind the observed transformations, including the synthetic property changing.

For example, in time $t_1$ CALM activates the schema $\Xi_1 = (\#0 + c \rightarrow \#1)$, where the context and expectation are composed by two elements (the first one synthetic and the second one perceptive), and one action. Suppose that the next situation '*#1'* is ambiguous, because it excites both schemas $\Xi_2 = (\clubsuit 1 + c \rightarrow \#0)$ and $\Xi_3 = (\blacklozenge 1 + c \rightarrow \#1)$. At this time, the mechanism cannot know the synthetic element value, crucial to determine what is the real situation. Suppose that, anyway, the mechanism decides to execute the action '*c'* in time $t_2$, and it is followed by the sensory perception '*0'* in $t_3$. Now, in $t_3$, the agent can deduce that the situation really dealt with in $t_2$ was '$\clubsuit 1$', and it can include this information into the schema activated in $t_1$, in the form $\Xi_1 = (\#0 + c \rightarrow \clubsuit 1)$.

## 4 Experiment

The CALM mechanism has already been used to successfully solve problems such as *flip*, which is also used by (Singh et al., 2003) and (Holmes; Isbell, 2006), and *wepp*, which is an interesting RL situated problem. CALM is able to solve both of them by creating new synthetic elements to represent underlying states of the problem (Perotto, 2010), (Perotto; Álvares, 2007), (Perotto et al., 2007). In this paper, we

introduce an experiment that we call *meph* (acronym to the actions that the agent can do: move, eat, play, hit).

In the *meph* experiment, the agent is inserted into a bi-dimensional environment (like a "grid") where it should learn how to interact with other agents and objects that can be found during its life. Our agent needs to create a policy of actions in a way to optimize its feelings (internal rewards). When it is hungry, it feels good by eating some food (which is an object that can be sometimes found in the environment, among other non eatable objects like stones). Agents and objects can be differentiated by observable features, it means that our agent can sensorially distinguish what is the "thing" with which it is interacting. However, both food and stone have the same appearance, and in this way, to discover if the object is either food or stone, the agent needs to experiment, it means, the agent needs to explicitly experiment the environment in order to take more information about the situation, for example, hitting the object to listen what sound it makes. Figure 8 shows a random configuration of the environment.

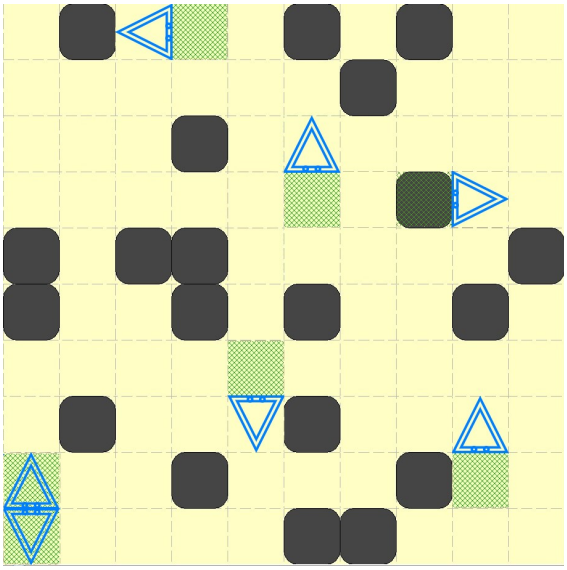

Figure 8: the simulation environment to the *meph* experiment, where the triangles represent the agents (looking and hearing forward), and the round squares represent the objects (food or stones).

When our agent is excited, it founds pleasure by playing with another agent. However, the other agents also have internal emotional states; when another agent is angry, any tentative to play with results in an aggression, which causes a disagreeable painful sensation to our agent. Playing is enjoyable and safe only if both agents are excited, or at least if the other agent is not angry (and then, not aggressive), but these emotional states are internal to each agent, and cannot be directly perceived. With such configuration, our agent will need to create new synthetic elements to be able to distinguish what is food and what is stone, and to distinguish who are aggressive and who are peaceable.

At each time step, our agent can execute one of 4 actions: *move*, *eat*, *play* or *hit*. Moving itself is a good strategy to scape from other aggressive agents, but it is also the action that changes the context, allowing the agent to search for different contexts. The agent does not control precisely the movement it does; the action of moving itself causes a random rotation followed by a position changing to an adjacent cell. Eating is the good choice when the agent is hungry and it is in front of some food at the same time, action that ceases the bad sensation caused by hungry. Playing is the good action to be carried out when the agent is excited and in frontal contact with another non-aggressive agent. Hitting is the action that serves to interacting with other objects without compromise; doing it, the agent is able to identify the solution to some ambiguous situations; for example, hitting a stone has no effect, while hitting food provokes a funny sound. The same for another agents, that reacts with a noisy sound when hit, but only if it is already angry.

The agent has two limited external perceptions, both focused on the cell that is directly in front of it; the sense of *vision* allows it to see if the place before him contains an object, an agent, or nothing; the sense of *hearing* permits to listen the sounds coming from there. The agent's body has 5 internal properties, corresponding to 5 equivalent internal perceptions: *pain*, *anger*, *hungry*, *excitation*, and *pleasure*. Pleasure occurs always when the agent plays with another agent, independently of the other agent internal state (which is quite selfish). However, as we know, our agent can get punched if the other agent is angry, and in this case pain takes place. When our agent feels pain and hungry at the same time, it becomes angry too. Initially the agent does not know anything about the environment or about its own

sensations. It does not distinguish the situations, and also does not know what consequences its actions imply.

The problem becomes interesting because playing can provoke both positive and negative rewards, the same for eating, that is an interesting behavior only in certain situations; it depends on the context where the action is executed, which is not fully observable by sensors. This is the model that CALM needs to learn by itself before establish the behavior policy. Figure 9 shows the involved variables, which will compose the schemas' identifying vectors. Figures 10 to 17 shows the anticipation trees created by the mechanism after stabilization.

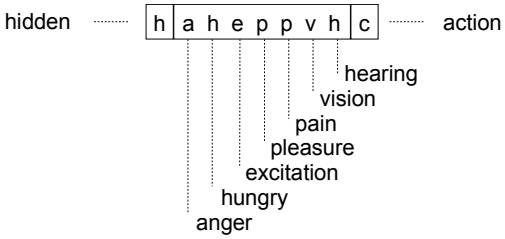


Figure 9: the vectors that compose the context of a schema; synthetic properties, perceptive properties, and controllable properties (actions).

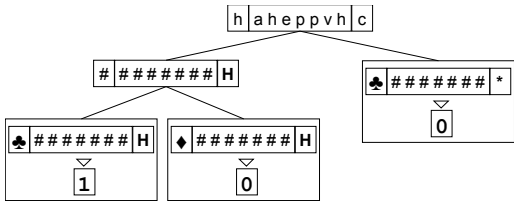


Figure 10: anticipation tree for hearing; the only action that provokes sound is hitting, it is true only if the object is food or the agent is hungry; when the agent hits (H) an object that is food or a non-aggressive agent, differentiated of their confounding pairs by the synthetic element (♣), then the agent listens a sound in the next instant; if the action is other (*) than hitting, then no sound is produced.

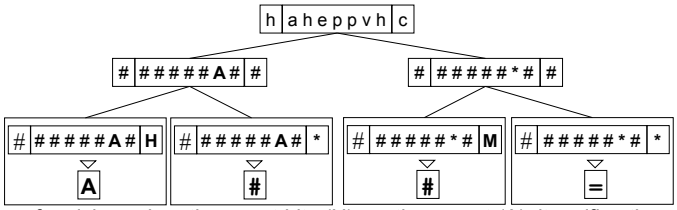


Figure 11: anticipation tree for vision; when the agent hits (H) another agent (A), it verifies the permanence of this other agent in its visual field, which means that hitting an agent makes it stay in the same place; however, no other actions (*) executed in front of an agent (A) can prevents it to go, and so the prediction is undefined (#); the same for moving itself (M), which causes a situation changing and the non predictability of the visual field for the next instant; in all of the other cases the vision stays unchanged.

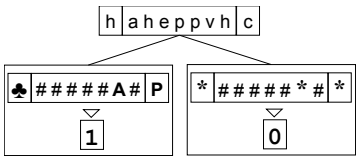


Figure 12: anticipation tree for pain; the agent knows that playing (P) with an aggressive (♣) agent (A) causes pain; otherwise, no pain is caused.

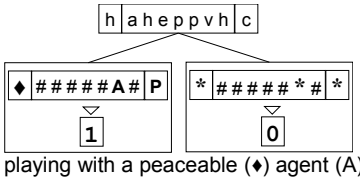


Figure 13: anticipation tree for pleasure; playing with a peaceable (♦) agent (A) is pleasant, and it is the only known way to reach this feeling.

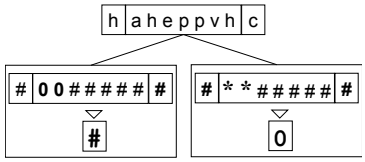

Figure 14: anticipation tree for excitation; when the agent feels neither anger (0) nor hungry (0), it can (eventually) becomes excited; it happens in a non-deterministic way, and for this reason the prediction is undefined in this case (#), which can be understood as a possibility; otherwise (**) excitation will be certainly absent.

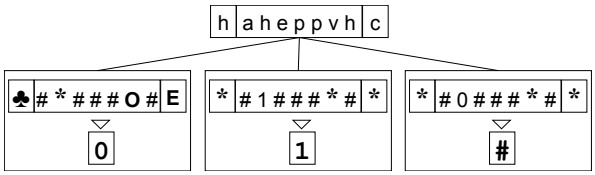


Figure 15: anticipation tree for hungry; eating (E) food (O♣) ceases hungry; otherwise, if the agent is already hungry, it will remains, but if it is not yet hungry, it can become.

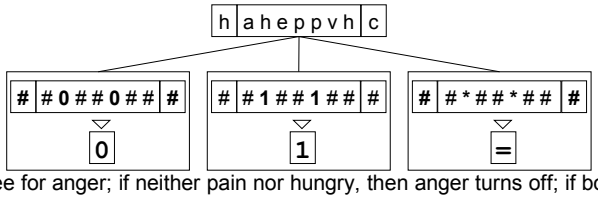


Figure 16: anticipation tree for anger; if neither pain nor hungry, then anger turns off; if both pain and hungry, then anger turns on; otherwise, anger does not change its state.

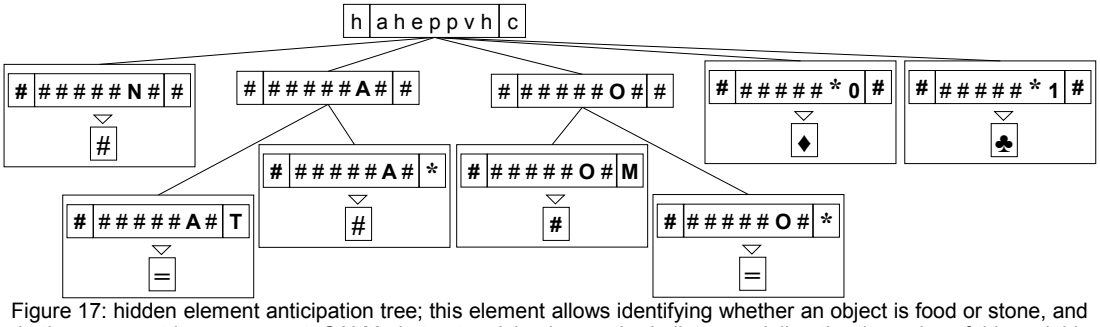


Figure 17: hidden element anticipation tree; this element allows identifying whether an object is food or stone, and whether an agent is angry or not; CALM abstract anticipation method allows modeling the dynamics of this variable, even if t is not directly observable by direct sensory perception; the perception of the noise (1) is the result that enables the discovering of the value of this hidden property; the visual perception of an object (O), or the fact of hitting (H) another agent (A), also permit to know that the hidden element does not change.

Figure 18 shows the evolution of the mean reward comparing the CALM solution with a random agent, and with two classic *Q-Learning* (Watkins; Dyan, 1992) implementations: the first one non situated (the agent has the vision of the entire environment as flat state space), and the second one with equivalent (situated) inputs than CALM.

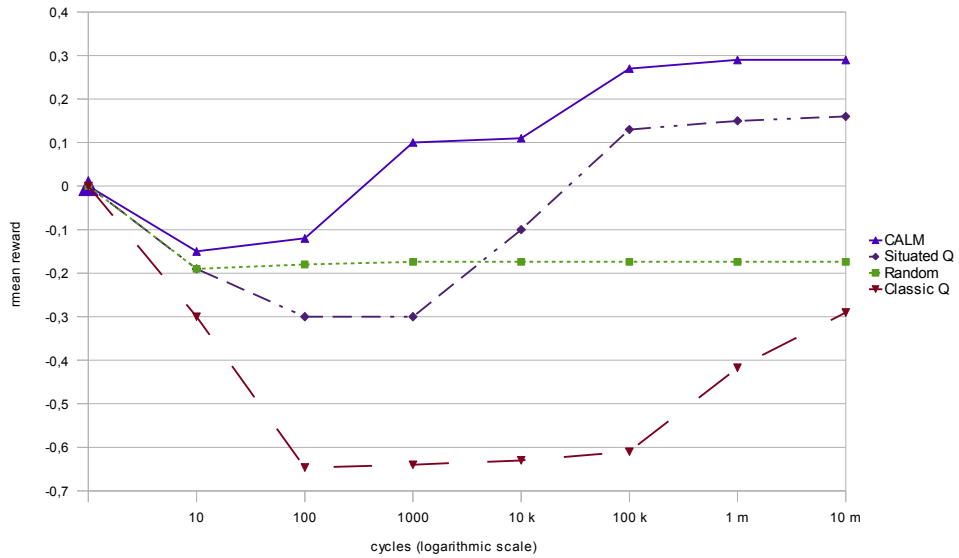


Figure 18: the evolution of mean reward in a typical execution of the *meph* problem, considering four different agent solutions: CALM, situated Q-Learning, Random, and Classic Q-Learning; the scenario is a 25x25 grid, where 10% of the cells are stones and 5% are food, in the presence of 10 other agents.

The non-situated implementation of Q-Learning algorithm (Classic Q) takes much more time to start drawing a convergence curve than the others, and in fact, the expected solution will never be reached; it is due to the fact that Q-Learning tries to construct directly a mapping from states to actions (a policy), but the state space is taken as the combination of the state variables; in this implementation (because it is not situated) each of the cells in the environment compose a different variable, and then the problem becomes quickly big; by the same cause, the agent becomes vulnerable to the growing of the board (the grid dimensions imply directly in the complexity of the problem).

The CALM solution converges much earlier than Q-Learning, even taken in its situated version, and CALM found a better solution also; it is due to the fact that CALM quickly constructs a model to predict the environment dynamics, including the non-observable properties in the model, and so it is able to define a good policy sooner. The "pause" in the convergence that can be seen in the graphic indicates two moments: first, the solution found before correctly modeling the hidden properties as synthetic elements, and then, the solution after having it. On the other side, Q-Learning stays attached to a probabilistic model, and in this case, without the information about the internal states of the other agents, trying to play with them becomes unsafe, and the Q-Learning solution will prefer do not do it.

## 5 Conclusions

The CALM mechanism, presented in (Perotto, 2010), can provide autonomous adaptive capability to an agent, because it is able to incrementally construct knowledge to represent the deterministic regularities observed during its interaction with the system, even in partially deterministic and partially observable environments. CALM can deal with the incomplete observation trough the induction and prediction of hidden properties, represented by synthetic elements, thus it is able to overpass the limit of sensory perception, constructing more abstract terms to represent the system, and to describe its dynamics in more complex levels. CALM can be very efficient to construct a model in non-deterministic environments if they are well structured. In other words, if the most part of transformations are in fact deterministic relatively to the underlying partially observable properties, and if the interdependence between variables are limited to a small range. Several problems found in the real world present these characteristics.

The proposed experiment (*meph*) can be taken as an useful problem that, even if simple, challenges the agent to solve some intricate issues such as the interaction with other complex agents. The next step in this way is to insert several cognitive agents like CALM in the same scenario; it means, agents that change our own internal model and policy of action, and in this way, present a non-stationary behavior. Again, the difficulty for one agent model the other agents in this kind of condition is even harder.

Finally, we believe that the same strategy can be adapted to several kinds of classification tasks, where a previous database of samples are available. In this case, the algorithm learns to classify new instances based on a model created from a training set of instances that have been properly labeled with the correct classes. This task is similar to several real world problems actually solved with the computer aim, such as e-mail filtering, diagnostic systems, recommendation systems, decision support systems, and so on.

## References


Astington, J.W., Harris, P.L., Olson, D.R. (eds.). (1988). **Developing Theories of Mind**, New York: Cambridge University Press, Cambridge.

Astrom, K.J. (1965). *Optimal Control of Markov Decision Processes with Incomplete State Estimation*. **Journal of Mathematical Analysis and Applications**, v.10, pp.174-205.

Bateson, G. (1972). **Steps to an Ecology of Mind**: *Collected Essays in Anthropology, Psychiatry, Evolution, and Epistemology*. University Of Chicago Press.

Beer, R.D. (1995). *A dynamical systems perspective on agent-environment interactions*. **Artificial Intelligence**. Elsevier, v.72, pp.173-215.

Bellman, R. (1957). *A Markovian Decision Process*. **Journal of Mathematics and Mechanics**, v.6.

Blum, A.L.; Langley, P. (1997). *Selection of relevant features and examples in machine learning*. **Artificial Intelligence**, Elsevier, v.97, pp.245-271.

Blythe, J. (1999). *Decision-Theoretic Planning*, **AI Magazine**, AAAI, v.20, n.2, pp.37-54.

Boutilier, C.; Poole, D. (1996). *Computing optimal policies for partially observable decision processes using compact representations*. In: National Conference on Artificial Intelligence, AAAI, 13rd, 1996, Portland, OR, USA. **Proceedings…** Portland: AAAI Press, v.2. pp.1168-1175.

Boutilier, C.; Dearden, R.; Goldszmidt, M. (2000). *Stochastic dynamic programming with factored representations*. **Artificial Intelligence**, Elsevier, v.121, n.1-2, pp.49-107.

Chrisman, L. (1992). *Reinforcement Learning with Perceptual Aliasing: the perceptual distinctions approach*. In: National Conference on Artificial Intelligence, AAAI, 10th, 1992, San Jose, CA, USA. **Proceedings…** AAAI Press. pp.183-188.

Dean, T.; Kanazawa, K. (1989). *A model for reasoning about persistence and causation*. **Comp. Intel.**, v.5, n.3, pp.142-150.

Degris, T.; Sigaud, O. (2010). *Factored Markov Decision Processes*. In: Buffet, O; Sigaud, O. (eds.). **Markov Decision Processes in Artificial Intelligence**. Vandoeuvre-lès-Nancy: Loria.

Degris, T.; Sigaud, O.; Wuillemin, P-H. (2006). *Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems*. In: International Conference on Machine Learning. 23th ICML. **Proceedings**, Pittsburgh, Pennsylvania: ACM, pp.257-264.

Dennett, D. (1987). **The Intentional Stance**. Cambridge, MA: MIT Press.

Drescher, G. (1991). **Made-Up Minds: a constructivist approach to artificial intelligence**. Cambridge, MA: MIT Press.

Feinberg, E.A.; Shwartz, A. (2002). **Handbook of Markov Decision Processes**: methods and applications. Norwell: Kluwer.

Friedman, N.; Koller, D. (2003). *Being Bayesian about Network Structure: a bayesian approach to structure discovery in bayesian networks*. **Machine Learning**, v.50, n.1-2, pp.95-125.

Guestrin, C.; Koller, D.; Parr, R. (2001). *Solving Factored POMDPs with Linear Value Functions*. In: Workshop on Planning under Uncertainty and Incomplete Information, 2001, Seattle, WA. **Proceedings…** pp.67-75.

Guestrin, C.; Koller, D.; Parr, R.; Venkataraman, S. (2003). *Efficient Solution Algorithms for Factored MDPs*. **Journal of Artificial Intelligence Research**. AAAI Press, v.19, pp.399-468.

Hansen, E.A.; Feng, Z. (2000). *Dynamic programming for POMDPs using a factored state representation*. In: International Conference on Artificial Intelligence, Planning and Scheduling, AIPS, 5th, 2000, Breckenridge, CO, USA. **Proceedings…** AAAI Press. pp.130-139.

Hauskrecht, M. (2000). *Value-function approximations for partially observable Markov decision processes*. **Journal of Artificial Intelligence Research**, AAAI Press, v.13, pp.33-94.

Holmes, M.; Isbell, C. (2006). *Looping Suffix Tree-Based Inference of Partially Observable Hidden State*. In: International Conference on Machine Learning. 23th ICML. **Proceedings**, Pittsburgh, Pennsylvania: ACM. pp.409-416.

Jensen, F.B. Graven-Nielsen, T. (2007). **Bayesian Networks and Decision Graphs**. (2.ed.). Springer.

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

Howard, R.A.; Matheson, J.E. (1981). **Influence Diagrams**. In: *The Principles and Applications of Decision Analysis*, p.720-762.

Hoey, J.; St-Aubin, R.; Hu, A.J.; Boutilier, C. (1999). *SPUDD: Stochastic Planning Using Decision Diagrams*. In: International Conference on Uncertainty in Artificial Intelligence, UAI, 15th, 1999, Stockholm, Sweden. **Proceedings…** San Francisco, CA: Morgan Kaufmann.

Jonsson, A.; Barto, A. (2005). *A Causal Approach to Hierarchical Decomposition of Factored MDPs*. In: International Conference on Machine Learning, ICML, 22nd, 2005, Bonn, Germany. **Proceedings…** ACM. pp.401-408.

Kaelbling, L.P.; Littman, M.L.; Cassandra, A.R. (1998). *Planning and acting in partially observable stochastic domains.* **Artificial Intelligence**, Elsevier, v.101, pp.99-134.

Kaelbling, L.P.; Littman, M.L.; Cassandra, A.R. (1994). *Acting optimally in partially observable stochastic domains.* In: National Conference on Artificial Intelligence, AAAI, 12th, 1994, Seattle, WA, USA. **Proceedings…** AAAI Press. pp.1023-1028.

Meuleau, N.; Kim, K-E.; Kaelbling, L.P.; Cassandra, A.R. (1999). *Solving POMDPs by Searching the Space of Finite Policies*. In: International Conference on Uncertainty in Artificial Intelligence, UAI, 15th, 1999, Stockholm, Sweden. **Proceedings…** San Francisco, CA: Morgan Kaufmann. pp.427-443.

Murphy, G.L. (2002). **The big book of concepts**. Cambridge, MA: MIT Press.

Pearl, J. (2000). **Causality**: models of reasoning and inference. Cambridge University Press.

Perotto, F.S.; Álvares, L.O. (2007). *Incremental Inductive Learning in a Constructivist Agent*. In: Research and Development in Intelligent Systems XXIII, **SGAI-2006 Proceedings**. London: Springer-Verlag, pp.129-144.

Perotto, F.S.; Álvares, L.O.; Buisson, J.-C. (2007). Constructivist Anticipatory Learning Mechanism (CALM): Dealing with Partially Deterministic and Partially Observable Environments. In: International Conference on Epigenetic Robotics, 7th, 2007, Piscataway, NJ, USA. **Proceedings…** New Jersey: Lund University Cognitive Studies. pp.117-127.

Perotto, F.S. (2010). **Un Mécanisme Constructiviste d'Apprentissage Automatique d'Anticipations pour des Agents Artificiels Situés**. PhD Thesis. Toulouse, France: INP. (in french)

Piaget, J. (1947). **La Psychologie de l'Intelligence**. Paris: Armand Colin.

Poupart, P.; Boutilier, C. (2004). *VDCBPI: an approximate scalable algorithm for large scale POMDPs*. In: Advances in Neural Information Processing Systems, NIPS, 17th, 2004, Vancouver, Canada. **Proceedings…** Cambridge: MIT Press. pp.1081-1088.

Puterman, M.L. (1994). **Markov Decision Processes**: discrete stochastic dynamic programming. New York: Wiley.

Shani, G.; Brafman, R.I.; Shimony, S.E. (2005). *Model-Based Online Learning of POMDPs*. In: European Conference on Machine Learning, ECML, 16th, 2005, Porto, Portugal. **Proceedings…** Berlin: Springer-Verlag. pp.353-364. (LNCS 3720).

Shani, G.; Poupart, P.; Brafman, R.I.; Shimony, S.E. (2008). *Efficient ADD Operations for Point-Based Algorithms*. In: International Conference on Automated Planning and Scheduling, ICAPS, 8th, 2008, Sydney, Australia. **Proceedings…** AAAI Press. pp.330-337.

Sim, H.S.; Kim, K.-E.; Kim, J.H.; Chang, D.-S.; Koo, M.-W. (2008). *Symbolic Heuristic Search Value Iteration for Factored POMDPs*. In: National Conference on Artificial Intelligence, AAAI, 23rd, 2008, Chicago, IL, USA. **Proceedings…** AAAI Press. pp.1088-1093.

Singh, S.; Littman, M.; Jong, N.; Pardoe, D; Stone, P. (2003). *Learning Predictive State Representations*. In: International Conference on Machine Learning, ICML, 20th, 2003, Washington, DC, USA. **Proceedings…** AAAI Press. pp.712-719.

Smallwood, R.D.; Sondik, E.J. (1973). *The optimal control of partially observable Markov decision processes over a finite horizon*. **Operations Research**, Informs, v.21, pp.1071-1088.

St-Aubin, R.; Hoey, J.; Boutilier, C. (2000). *APRICODD: Approximate policy construction using decision diagrams*. Advances in Neural Information Processing Systems, NIPS, 12nd, 1999, Denver, CO, USA. **Proceedings…** Cambridge: MIT Press.

Strehl, A.L., Diuk, C., Littman, M.L. (2007). *Efficient Structure Learning in Factored-State MDPs*. In: National Conference on Artificial Intelligence, AAAI, 22nd, 2007, Vancouver, Canada. **Proceedings…** AAAI Press. pp.645-650.

Suchman, L.A. (1987). **Plans and Situated Actions**. Cambridge University Press.

Sutton, R.S.; Barto, A.G. (1998). **Reinforcement Learning**: an introduction. MIT Press.

Watkins, C.J.C.H.; Dayan, P. (1992). *Q-learning*. **Machine Learning**, v.8, n.3, pp.279-292.

Wilson, R.; Clark, A. (2008). *How to Situate Cognition: Letting Nature Take its Course.* In: Aydede, M.; Robbins, P. (eds.). **Cambridge Handbook of Situated Cognition**. New York: Cambridge University Press.