Improving MACS thanks to a comparison with 2TBNs

Olivier Sigaud, Thierry Gourdin and Pierre-Henri Wuillemin

Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie - Paris6
4 place Jussieu, Paris, F-75005 France

Abstract. Factored Markov Decision Processes is the theoretical framework underlying multi-step Learning Classifier Systems research. This framework is mostly used in the context of Two-stage Bayes Networks, a subset of Bayes Networks. In this paper, we compare the Learning Classifier Systems approach and the Bayes Networks approach to factored Markov Decision Problems. More specifically, we focus on a comparison between MACS, an Anticipatory Learning Classifier System, and Structured Policy Iteration, a general planning algorithm used in the context of Two-stage Bayes Networks. From that comparison, we define a new algorithm resulting from the adaptation of Structured Policy Iteration to the context of MACS. We conclude by calling for a closer communication between both research communities.

1 Introduction

As [Lan02] did show very clearly, Learning Classifier Systems (LCSs) are a family of Reinforcement Learning (RL) systems endowed with a generalization property. The usual formal representation of RL problems is a Markov Decision Process (MDP), which is defined by a finite state space S, a finite set of actions A, a transition function T and a reward function R.

In the LCS framework, the states consist of several independent attributes whose value define a perceptual situation. Indeed, instead of the (state, action, payoff) triples in the Q-Table of standard RL algorithms, a LCS contains rules called "classifiers" organized into a `[condition]` part (or `C` part), an `[action]` part (or `A` part) and a payoff. The `C` part specifies the value of a set of attributes. It is satisfied only if all values specified match the corresponding values in the current state.

In the LCS framework, finding a compact representation is seen as a generalization problem. The generalization capability of LCSs comes from their use of *don't care* symbols `#` in the `C` part of the classifiers. Indeed, a `#` *matches* any particular value of the considered attribute. Therefore, changing an attribute into a `#` makes the corresponding `C` part more general (it matches more situations). As a result, the representation is more compact.

The same framework is also used in a sub-part of the Bayes Networks (BNs) community, under the name "factored MDP". Indeed, a factored MDP is a MDP where each state is a collection of random variables. This representation takes advantage on local probabilistic conditional dependences among some variables to build a more compact model of the underlying MDP by factoring the dependent variables. More precisely, according to [BDH99], if the situation is composed of n variables such that

any of them only depends of the value of l other variables at the previous time step, then the size of the factored representation is in $O(n2^l)$ instead of $O(2^n)$.

Thus the intuition giving rise to the use of factored MDPs in BNs seem to be exactly the same as the intuition underlying LCS research, particularly in the multi-step context. In both cases, the goal is to solve MDPs with a compact representation.

Given this identity, we present a comparison between both perspectives, showing that 2TBN, a formalism devoted to solving factored MDPs in BNs, is very similar to the one of MACS, an Anticipatory Learning Classifier System (ALCS). Then, as a result of the comparison, we show how the Structured Policy Iteration (SPI) algorithm designed in the 2TBN case can be adapted to MACS and we discuss the improvements that result from this adaptation.

The paper is organized as follows. In section 2, we briefly present the formalism used in MACS. In section 3, we present how 2TBNs tackle factored MDP problems and we illustrate their formalism through an example used as a basis for a comparison. In section 4, we compare the 2TBN formalism with the one used in MACS, and in section 5, we show how to adapt SPI to MACS. We discuss in section 6 the insights that result from this adaptation both by the 2TBN side and by the ALCS side. Finally, we conclude by calling for more exchanges between both research communities.

2 Brief overview of MACS

2.1 Anticipatory Learning Classifier Systems

As indicated in the introduction, among RL systems, LCSs are a family of systems designed to solve problems where the state consists of several attributes. They take advantage of factored representations with respect to plain RL techniques tabular *Q-learning* thanks to their generalization capability. As a consequence, they can solve problems with fewer classifiers than a system dealing with tabular representations.

Most standard LCSs call upon a combination of a RL algorithm such as *Q-learning* with a Genetic Algorithm (GA) [Gol89]. The RL algorithm estimates the quality of actions in the situations specified by the different classifiers, while the GA evolves the population of classifiers. Thus each classifier holds both an estimated quality and a fitness. In *strength-based* LCSs such as ZCS [Wil94], the fitness is equal to the estimated quality. As a result, only the classifiers specifying actions with a high payoff are kept. In *accuracy-based* LCSs such as XCS [Wil95], the fitness is equal to the capacity of the classifier to accurately predict the payoff it will receive. This results in a more efficient coverage of the (state, action) space, and XCS is now the most widely used LCS. For a general overview of recent LCS research, see [LR00].

Instead of directly learning a model of the quality function as standard LCSs do, Anticipatory Learning Classifier Systems (ALCSs) such as ACS [Sto98,BGS00], ACS2 [But02], YACS [GS01,GSS02] and MACS [GMS03,GS03] learn a model of the transition function T. This model is then used to speed up the RL process.

In ALCSs, the classifiers are organized into `[condition] [action] [effect]` parts, denoted `C-A-E`. The `E` part represents the effects of action `A` in situations matched by condition `C`. It records the perceived changes in the environment. Different formalisms giving rise to generalization in this approach are described hereafter.

2.2 From YACS to MACS

In ACS, ACS2 and YACS, a `C` part may contain *don't care* symbols "`#`" or specific values (like 0 or 1), as in XCS. An `E` part may contain either specific values or *don't change* symbols "`=`", meaning that the attribute remains unchanged when the action is performed. A specific value in the `E` part means that the value of the corresponding attribute *changes* to the value specified in that `E` part.

Unfortunately, such a formalism cannot represent regularities across different attributes from one time step to another, though such regularities are common.

[01020100]	[Action]	*Anticipated situation*
[0#######]	[East]	[???????0]
[#1######]	[East]	[1???????]
[##02####]	[East]	[????2???]
[###2####]	[East]	[????1???]
[####0###]	[East]	[?????0??]
[#####1##]	[East]	[?1??????]
[######0#]	[East]	[??0?????]
[#######0]	[East]	[???1????]
[########]	[East]	[??????0?]
Resulting anticipations	→	[11012000]
	or	[11011000]

Table 1. During the integration process, MACS scans the `E` parts and selects classifiers whose `A` part matches the action and whose `C` part matches the situation. The integration process builds all the possible anticipated situations with respect to the possible values of every attribute. Here, MACS anticipates that using `[01020100]` as a current situation should lead either to `[11012000]` or to `[11011000]`. If all the classifiers were accurate, this process would generate only one possible anticipation.

In order to discover more regularities, our second ALCS, MACS [GMS03,GS03], uses *don't know* symbols "`?`" instead of *don't change* symbols in the `E` part. Thanks to that formalism, the overall system gains the opportunity to discover regularities across different attributes in the `C` and the `E` parts. As a result of this modification, we can split the model of transitions into modules predicting the value of different attributes. The consequence is that the system needs an additional mechanism which *integrates* the partial anticipations provided by the modules and builds a whole anticipated situation, without any *don't know* symbol in its description, as shown in Table 1.

2.3 The architecture of MACS

The MACS architecture is illustrated in figure 1. As shown in [GS03], it is similar to a DYNA architecture in all respects, but it is additionally endowed with a generalization capability. It consists of a model of transitions, which models the dynamics of the agent-environment interactions, and a model of the payoff, which models the payoffs that the agent can get from its actions. The main processes in MACS are devoted to learning the model of transitions thanks to dedicated generalization and specialization heuristics, and learning the model of the payoff, according to different criteria that endow MACS with an active exploration capability. These processes have already

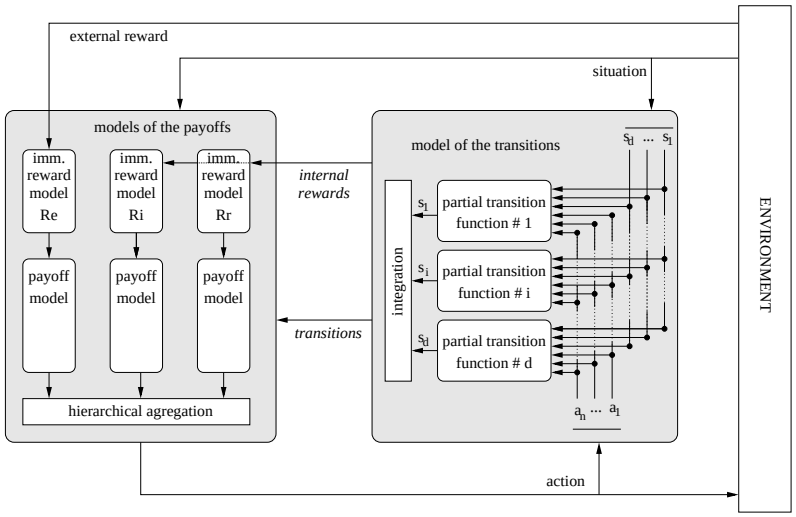


Fig. 1. MACS global architecture.

been presented in detail in [GMS03]. We must just mention that MACS uses the model of transitions to speed up the learning of the model of the payoff thanks to a Value Iteration algorithm that is somewhat inefficient: the model of the payoff consists in storing two values for each encountered state without generalization, and Value Iteration requires a costly lookahead operation involving the anticipations integration process described in section 2.2.

Experimental results presented in [GMS03] demonstrated that the new formalism used by MACS actually affords more powerful generalization capacities than the former ones, without any cost in terms of learning speed.

3 DBNs and 2TBNs

As we have said in the introduction, a factored MDP is a MDP where all states are defined as collections of random variables X_i. The set of states can be denoted $S = (X_1, ..., X_n)$. Each variable X_i takes its value x_i in a domain: $\forall i, x_i \in Dom(X_i)$. Thus a given state s is defined by a vector of values of the random variables: we have $s = (x_1, ..., x_n)$ which defines the states exactly as in LCSs. A Bayesian Network (BN) is a tool to deal with probabilistic dependencies among these random variables.

The BN framework [Pea88] includes a graphical formalism devoted to the representation of conditional relations between variables. Graphically, variables are represented as nodes in a Directed Acyclic Graph. A link between two nodes represent a probabilistic dependency between the corresponding variable such that the joint probability of all variables can be decomposed into a product of conditional probabilities of each variable, given its parents in the graph. Hence BNs provide an easy way to specify conditional independences between variables. Moreover, they provide a compact parameterization of the model.

Dynamic Bayesian Networks (DBNs) [DK89] are BNs of stochastic processes, representing (temporal) sequential data. They extend BNs by focusing on modeling

changes on stochastic variables ($X_{t+1} = f(X_t)$) over time with a BN. Hence the sequence is infinite but the time slice is finite and (usually) static. DBNs generalize Hidden Markov Models (HMMs), Linear Dynamical Systems (LDSs) and Kalman filters, and represent (hidden and observed) states in terms of probabilistic variables. Note that the assumption of time slice invariability (relations do not change over time) may be relaxed for the parameters as well as for the structure. Usually, a DBN is represented by 2 time slices, under the name 2TBN (2 Time (slices) Bayesian Network). The former slice represents the initial state of the model; the second one represents the generic state. The relations between the two slices represent the stochastic process. In the following we will consider restrictions of 2TBNs where there are no relations between variables in the second time slice.

With respect to the standard BNs, this particular class is very restricted, but it is nevertheless rich enough to adequately represent factored MDPs and give rise to tractable algorithms, whereas more powerful representations are generally untractable.

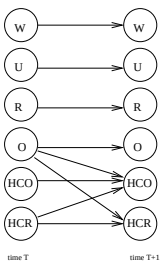
In order to show how 2TBNs are used to model a factored MDP, we will use the example given in [BDG00]. In this example, a robot must go to a café to buy some coffee and deliver it to its owner in her office. It may rain on the way, in which case the robot may get wet, unless it has an umbrella. There are six boolean propositions describing the state of the system:

- O: the robot is at the office ($\overline{O}$ means that it is at the café);
- R: it is raining; W: the robot is wet;
- U: the robot has its umbrella;
- HCR: the robot has coffee; HCO: the owner has coffee.

The robot can take four actions:

- Go: move to the opposite location; $GetU$: get the umbrella.
- $BuyC$: buy coffee; $DelC$: deliver coffee to the user;

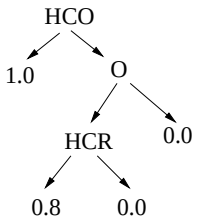
A 2TBN can be used to model the effect of each action on the state of the problem. Such 2TBNs are called "action networks" [DG94]. In figure 2.a, we show the action network corresponding to the action "DelC".



(a)

O^t	HCR^t	HCO^t	HCO^{t+1}
t	t	t	1.0
f	t	t	1.0
t	f	t	1.0
f	f	t	1.0
t	t	f	0.8
f	t	f	0.0
t	f	f	0.0
f	f	f	0.0

(b)



(c)

Fig. 2. (a) The action network for $DelC$, represented as a 2TBN; (b) The CPT for $HCO/DelC$; (c) The corresponding tree representation

From this network we can see for instance that, when the robot delivers coffee, the fact that he is wet afterwards does only depend on whether he was wet beforehand,

since delivering coffee does not happen outdoors. On the contrary, the fact that the owner finally gets some coffee depends on three propositions: is the robot at the office? did the robot have coffee? did the owner have coffee?

There is one such network per action. But these networks do only indicate whether there is a probabilistic dependency or not, they do not provide enough information to determine the actual dynamics of the environment. The missing information is represented in Conditional Probabilistic Tables (CPTs). An example of CPT is shown in figure 2.b. This table gives the probability that HCO will hold depending on O, HCR and HCO if the agent takes action $DelC$. Finally, the same information can be represented in a tree, as shown in figure 2.c. One such table or tree must be given for each variable and for each action in order to get a complete specification of the dynamics of the factored MDP.

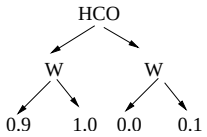


Fig. 3. The reward tree for the café example

Finally, one can also represent the reward function of the problem with a decision, tree, as shown in figure 3. Let us now compare this formalism with the one used in MACS before comparing the processes used to deal with these representations.

4 Comparison of formalisms

First of all, at the level of the organization of data, the representations in 2TBNs and MACS are slightly different.

In MACS, each module in the model of transitions tries to predict the value of one particular attribute for all possible situations and all possible actions, while in the standard BN representation, there is one 2TBN for each possible action and then, for any such action, there is one CPT for each predicted attribute.

Thus in the 2TBN representation, the distinction between actions is prior to the distinction between attributes, while it is the contrary in MACS. But this is just a different organization for the same data.

At a lower level, the content of these data is similar. Indeed, for a given action, CPTs convey the same information as a subset of possible modules dedicated to predicting the same attribute for the same action in MACS, as shown in table 2.

Thus expressing a CPT as a set of MACS classifiers is straightforward: for each leaf in the tree, add a new classifier with a `#` in front of all irrelevant variables in the `C` part, the considered action in the `A` part, and an `E` part whose only specialized attribute is the one predicted by the CPT.

When this is done, one obvious difference remains: the CPT conveys a probability that the attribute will take the predicted value, while MACS conveys a deterministic information. Indeed, MACS was designed to solve deterministic MDPs and should be extended in that direction if we want to reduce the distance between both formalisms.

W	U	R	O	HCR	HCO	Action	W	U	R	O	HCR	HCO	**missing proba.**
#	#	#	#	#	1	DelC	?	?	?	?	?	1	**1.0**
#	#	#	1	1	0	DelC	?	?	?	?	?	1	**0.8**
#	#	#	#	0	0	DelC	?	?	?	?	?	0	**1.0**
#	#	#	0	#	0	DelC	?	?	?	?	?	0	**1.0**

Table 2. The CPT for *HCO*/*DelC* represented with the formalism of MACS

Another important difference comes from the way the payoff is modeled in both approaches. In MACS we map the values to all situations encountered, giving rise to a flat (situation, value) table. On the contrary, [BDG95] indicates that the reward function can be represented as a tree whose nodes are variables, as shown in figure 3. This gives rise both to a more compact representation and to more efficient algorithms, as we will discuss now.

5 Incorporating a 2TBN planning algorithm in MACS

The main difference between mechanisms in MACS and in 2TBNs comes from the fact that MACS is designed to learn by itself a representation from its interaction with the MDP thanks to a RL mechanism, while in most 2TBNs cases the representation is considered as given by the user.

Thus MACS must solve both a learning problem and a planning problem while most 2TBNs just solve the planning problem. Note however that some authors are interested in learning the structure of DBNs [FMR98,HGC95,Gha97], but in general they do so separately from tackling the planning problem, drawing inspiration from algorithms developed for Hidden Markov Models [RJ86], and they focus on more complex structures than 2TBNs.

As a consequence of this difference, in order to go further in the comparison, we must focus on the way the planning problem is solved both in MACS and in 2TBNs.

5.1 Structured Policy Iteration

The algorithm we will present in order to build an optimal policy in 2TBNs is Structured Policy Iteration (SPI) [BDG95,BDG00]. SPI is inspired from the tabular Modified Policy Iteration (MPI) algorithm [How71,PS78] improved in order to work exclusively on tree structures, which results in favorable cases in a significant reduction of the computational cost of the algorithm with respect to tabular MPI. For a more detailed presentation of SPI than the one given below, read [BDG00].

The central operation common to all dynamic programming and reinforcement learning algorithms is the propagation of values or qualities among the states of the MDP [1], according to the Bellman equation:

$$\forall s_i \in S, Q_a^V(s_i) = R(s_i, a) + \beta \sum_{s_j \in S} T(s_i, a, s_j) V(s_j) \quad (1)$$

The key idea in SPI consists in achieving this propagation more efficiently than in tabular representations by grouping together the states that share the same values

[1] This process is called "Decision-Theoretic Regression" in [BDG00].

into tree-based representations. Indeed, all structures in SPI are decision trees whose leaves correspond to groups of states sharing the same values on the variables present in the parent nodes of these leaves.

In this tree-based representation, we note:

- $QTree_a^V(s)$ the quality tree of action a given the value function V;
- $RTree(s,a)$ the tree of rewards obtained from performing action a in state s;
- $VTree(s)$ the value tree of state s.

As a consequence, the Bellman equation becomes:

$$\forall s_i \in S, QTree_a^V(s_i) = RTree(s_i,a) + \beta \sum_{s_j \in S} T(s_i,a,s_j) VTree(s_j) \qquad (2)$$

The value propagation algorithm in the tree-based case strictly follows the standard algorithm for the tabular case:

- It starts with an initial value function $VTree_0(s) = RTree(s,a)$.
- Then, in order to update $QTree_a^V(s)$, it looks for the term $T(s_i,a,s_j)$, i.e. for transitions from one state to the next through actions, so as to propagate the values along these transitions. Instead of looking for these transitions in a (s_t,a_t,s_{t+1}) table, the tree-based algorithm uses the CPTs of all actions, which directly give a tree-based representation of the same information.

This algorithm can infer the tree-based representation of the quality of all actions from a tree-based representation of the current value function and back, so as to perform a tree-based value iteration, that we will call Structured Value Iteration (SVI) hereafter. The details of this algorithm can be found in [BDG00].

Taking SVI as a basic component, SPI can be decomposed into two stages as for all Policy Iteration algorithms: a Structured Policy Evaluation [2] stage, and a Structured Policy Improvement stage. The former consists in evaluating the long term reward that an agent can expect from a given policy. It is based on the estimation of the value function explained above. The latter consists in improving the current policy according to the new value function calculated by the former. It can be seen as a variant of the former where, instead of labelling the leaves of $VTree(s)$ with the maximum estimated values, the algorithm labels them with the actions that deliver these maximum values.

5.2 CbVI, a Classifier-based SVI

As we briefly mentioned in section 2.3, in MACS, determining the reachable situations is an expensive process, since it implies the partial anticipations integration process described in section 2.2. This process is used both when the agent controlled by MACS chooses the next action and when the model of the payoff is improved thanks to a Value Iteration algorithm. Thus it would be interesting to improve this part of MACS by drawing inspiration from the way SVI is performed in 2TBNs.

In order to do so, we must distinguish two different sub-problems:

- **The planning problem**: Given a perfect model of transitions under the form of a list of classifiers and a perfect knowledge of the reward function, adapt SVI to work with classifiers. We deal with that sub-problem below.

[2] called Structured Successive Approximation (or SSA) in [BDG00].

– **The learning problem**: Adapt SVI or its classifier-based version to the case where the model of transitions is learned simultaneously with looking for an optimal policy. We keep this second sub-problem for the final discussion.

One way to integrate the insights from the SVI algorithm into MACS would be to derive from the classifier-based representation of the model of transitions a tree-based representation and then apply SVI as such.

In this section we rather present our adaptation of SVI to work directly with a classifier-based representation.

Let us call "token" the specification of a value or a `#` for one attribute, and let us call "message" a set of k tokens, where k is the number of attributes that define a state. In particular, the `C` part of classifiers are messages.

Our algorithm, called CbVI, heavily relies on an associative and commutative intersection operator $\cap$ between two messages. The intersection between two tokens T_a and T_b is defined as follows (we note x_a the value of token a):

$$T_a \cap T_b = \begin{cases} x_a & \text{if } x_a = x_b \text{ or } x_b = \# \\ x_b & \text{if } x_a = \# \\ \emptyset & \text{otherwise} \end{cases}$$

Now, if we note $Message = \{T_i(Message)\}_{i\in[1,m]}$ the fact that the message contains the tokens T_i for each attribute i, then we have:

$$Mess1 \cap Mess2 = \begin{cases} \emptyset \text{ if } \exists i \in [1,m] \text{ such that } T_i(Mess1) \cap T_i(Mess2) = \emptyset \\ \{T_i(Mess1) \cap T_i(Mess2)\}_{i\in[1,m]} \text{ otherwise} \end{cases}$$

With this operator, we can give the classifier-based version of SVI.

Let $\{PreL(A, T_i)\}$ be the list of `C` parts of classifiers in the model of transitions specifying action A and predicting T_i in their `E` part.

The current value function V_n is a list of p messages $(M_n)_{n\in[1,p]}$ with one reward value for each message. To compute V_{n+1}, we do the following:

1. For each action A
   (a) For each message M_n,
      i. For each value x_i of token $T_i(M_n)$ such that $x_i \neq \#$, retrieve $\{PreL(A, x_i)\}$.
      ii. Compute the list $L(A, M_n)$ of all the non-empty intersections between each possible combination of `C` parts from the different $\{PreL(A, T_i(M_n))\}$, taking one message in each list per $i \in \{1, ..., m\}$.
      iii. If some messages of $L(A, M_n)$ overlap, specialize them so as to get non-overlapping messages.
      iv. For each element M' in $L(A, M_n)$, compute the contribution of M_n to the value of M'. The attributes being considered independent, the probability of reaching M_n from M' is equal to the product of the probabilities given by each classifier implied in the intersection. We note Π_{proba} this product. Thus the contribution of M_n is $contribV(M_n, M') = \Pi_{proba} V(M_n)$.
   (b) If some messages M'_j and M'_k among the different $L(A, M_n), n \in [1, p]$ strictly overlap, specialize them so as to get non-overlapping messages M'' [3].

[3] If the messages defining V_n do not overlap, if the problem is markov and deterministic and if the `C` parts in the model of transitions do not overlap, there will not be any overlapping between the $L(A, M_n)$ lists, so this step can be simplified. But in a context where we learn the model of transitions and where new sources of rewards can be discovered, it is necessary to use this more secure version.

(c) For all M'' in all $L(A, M_n)$, sum the contributions from all states M_n computing their value according to the Bellman equation :

$$V(M'') = R(M'', A) + \beta \sum_{M_n} contribV(M_n, M'') \quad (3)$$

where $R(M'', A)$ is the immediate reward (if any) for performing action A in situations matched by M'' and β the discount factor. We obtain a unique list $L(A)$, corresponding to the quality function of the action A for any situation, according to the previous value function [4].

2. Finally, in order to compute V_{n+1}, find the maximum value for each M''' among the different $L(A)$ given by the different actions A. This implies the same merging process as in step 1c, taking the greatest value when two M''' overlap.

In step 2, we can record both the value of each situation and the action that gave rise to that value. By doing so, we record the policy giving the best action in any situation.

6 Discussion and Future Work

CbVI is very similar to SVI, apart from the fact that the basic operator in CbVI consists of an intersection between messages while SVI relies on *merging trees*, *appending trees* and *tree simplifications*. Though we did not measure that yet, we suspect that our version is much simpler to code, though computationally more expensive than the original SVI. Furthermore, CbVI overcomes two main drawbacks of MACS:

- it makes it possible to build a compact representation of the value function, where MACS previously had to store a value for each encountered state separately;
- as a side effect, it makes it possible to store a compact representation of the policy with very little extra cost, where MACS had to perform an expensive lookahead operation at each time step to select the best action.

Nevertheless, before incorporating CbVI into MACS, one must consider carefully the fact that MACS is a latent learning system building its own model of transitions while we made in the previous section the assumption that we had a perfect model of transitions and a perfect knowledge of the immediate reward function at hand.

To face the case where this assumption does not hold, there are four solutions:

- First build a perfect model of transitions and of immediate reward through random or active exploration before starting to perform value iteration loops. This is not satisfactory in the context of large problems.
- Reinitialize the value function to the immediate reward classifier-based representation each time the model of transitions or of immediate reward changes.
- Try to repair locally the current value function in case of model changes.
- Consider that CbVI will be robust and do nothing special. This is more or less what previous versions of MACS were doing, and it seems to work, so it must be investigated.

[4] In the deterministic case to which MACS was restricted so far, Π_{proba} is always 1.0 for one particular M_n and 0 for any other, thus this sum does not need to be computed.

The third solution seems to be the most appealing. On a more theoretical line of research, we believe that a formal functional equivalence between the SVI and CbVI can be proven, which would result in the possibility to import all the theoretical results from [BDG00] in our framework. This must be investigated soon.

7 Conclusions

As designers of MACS, we have drawn some interesting ideas about a way to make our system computationally more efficient by adapting the SVI algorithm to a classifier-based representation. But we feel that the reader can draw some more general lessons from the work presented in that paper.

Indeed, we believe that a closer communication between the LCS community and the BN community can result in a significant mutual enrichment, since both approaches come with a different perspective on the same framework, namely factored MDPs, and have developed different technical and theoretical tools to deal with that framework.

We feel that coming from one perspective to the other can raise interesting questions. For instance, since DBNs are just 2TBNs where some probabilistic dependencies between variables within the same time step (called "synchronic arcs") might hold, could such dependencies be integrated in the LCS framework without deeply reconsidering the classifier-based representation?

From a more global perspective, LCS researchers seem more focused on the learning problem than BN researchers, who are more interested in the planning/inference problem. It seems also that BN research is often more mathematically formalized than LCS research and that this formalization could be imported into the LCS framework.

Conversely, a greater knowledge of the LCS framework in the BN community would probably result in more interest in the 2TBN case, often disregarded as too simple by BN researchers. And, above all, the RL approach used in LCSs should probably be imported into the 2TBN context, since learning the structure of a problem is becoming an important research topic in the BN community.

References

[BDG95] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1104–1111, Montreal, 1995.

[BDG00] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1):49–107, 2000.

[BDH99] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[BGS00] M. V. Butz, D. E. Goldberg, and W. Stolzmann. Introducing a genetic generalization pressure to the Anticipatory Classifier System part I: Theoretical approach. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference (GECCO 2000)*, pages 34–41, 2000.

[But02] M. V. Butz. An Algorithmic Description of ACS2. In P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems*, volume 2321 of *LNAI*, pages 211–229. Springer-Verlag, Berlin, 2002.

[DG94] A. Darwiche and M. Goldszmit. Action networks: A framework for reasoning about action and change under uncertainty. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 136–144, Seattle, WA, 1994.

[DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[FMR98] N. Friedman, K. Murphy, and S. Russell. Learning the structure of dynamic probabilistic networks. In *Proceedings of UAI 98*, 1998.

[Gha97] Z. Ghahramani. Learning dynamic bayesian networks. In C. L. Giles and M. Gori, editors, *Adaptive Processing of Temporal Information*. LNAI, Springer-Verlag, Berlin, 1997.

[GMS03] P. Gérard, J.-A. Meyer, and O. Sigaud. Combining latent learning with dynamic programming. *European Journal of Operation Research*, to appear, 2003.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading MA, 1989.

[GS01] P. Gérard and O. Sigaud. YACS : Combining Anticipation and Dynamic Programming in Classifier Systems. In P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems*, volume 1996 of *LNAI*, pages 52–69. Springer-Verlag, Berlin, 2001.

[GS03] P. Gérard and O. Sigaud. Designing efficient exploration with macs: Modules and function approximation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2003 (GECCO03)*, pages 1882–1893, Chicago, IL, july 2003. Springer-Verlag.

[GSS02] P. Gérard, W. Stolzmann, and O. Sigaud. YACS: a new Learning Classifier System with Anticipation. *Journal of Soft Computing : Special Issue on Learning Classifier Systems*, 6(3-4):216–228, 2002.

[HGC95] D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[How71] R. A. Howard. *Dynamic Probabilistic Systems*. Wiley, 1971.

[Lan02] P.-L. Lanzi. Learning Classifier Systems from a Reinforcement Learning Perspective. *Journal of Soft Computing*, 6(3-4):162–170, 2002.

[LR00] P.-L. Lanzi and R. L. Riolo. A roadmap to the last decade of Learning Classifier Systems research (from 1989 to 1999). In P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems: from Foundations to Applications*, pages 33–62. Springer-Verlag, Heidelberg, 2000.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, San Mateo, CA, 1988.

[PS78] M. L. Puterman and M. C. Shin. Modified policy iteration algorithms for discounted Markov Decision Problems. *Management Science*, 24:1127–1137, 1978.

[RJ86] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, Jan.:4–16, 1986.

[Sto98] W. Stolzmann. Anticipatory Classifier Systems. In *Genetic Programming*, pages 658–664. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998.

[Wil94] S. W. Wilson. ZCS, a zeroth level Classifier System. *Evolutionary Computation*, 2(1):1–18, 1994.

[Wil95] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.