Anticipatory Learning Classifier Systems and Factored Reinforcement Learning

Olivier Sigaud[1], Martin V. Butz[4], Olga Kozlova[1,2], and Christophe Meyer[3]

[1] Université Pierre et Marie Curie - Paris6
Institut des Systèmes Intelligents et de Robotique (ISIR), CNRS FRE 2507,
4 place Jussieu, F-75005 Paris, France
Olivier.Sigaud@isir.fr
[2] Thales Security Solutions & Services, Simulation
1 rue du Général de Gaulle, Osny BP 226
F95523 Cergy Pontoise Cedex, France
Olga.Kozlova@thalesgroup.com
[3] Thales Security Solutions & Services, ThereSIS Research and Innovation Office
Route départementale 128
F91767 Palaiseau Cedex, France
Christophe.Meyer@thalesgroup.com
[4] University of Würzburg
Röntgenring 11
97070 Würzburg, Germany
mbutz@psychologie.uni-wuerzburg.de

Abstract. Factored Reinforcement Learning (FRL) is a new technique to solve Factored Markov Decision Problems (FMDPs) when the structure of the problem is not known in advance. Like Anticipatory Learning Classifier Systems (ALCSs), it is a model-based Reinforcement Learning approach that includes generalization mechanisms in the presence of a structured domain. In general, FRL and ALCSs are explicit, state-anticipatory approaches that learn generalized state transition models to improve system behavior based on model-based reinforcement learning techniques. In this contribution, we highlight the conceptual similarities and differences between FRL and ALCSs, focusing on the one hand on SPITI, an instance of FRL method, and on ALCSs, MACS and XACS, on the other hand. Though FRL systems seem to benefit from a clearer theoretical grounding, an empirical comparison between SPITI and XACS on two benchmark problems reveals that the latter scales much better than the former when some combination of state variables do not occur. Based on this finding, we discuss the mechanisms in XACS that result in the better scalability and propose importing these mechanisms into FRL systems.

1 Introduction

This paper is about two classes of explicit state-anticipatory systems [1] that learn generalized state transition models to improve their behavior based on model-based reinforcement learning techniques.

G. Pezzulo et al. (Eds.): ABiALS 2008, LNAI 5499, pp. 321–333, 2009.

On the one hand, Learning Classifier Systems (LCSs) are rule-based systems where the rules (called classifiers) are learned from experience. Due to genetic algorithm-based generalization mechanisms, LCSs were shown to build compact representations of Markov Decision Problems (MDPs) and learn to behave optimally. Anticipatory Learning Classifier Systems (ALCSs) [2] deviate from this classical framework on one fundamental point. Instead of [Condition] → [Action] classifiers, they manipulate [Condition] [Action] → [Effect] classifiers, where the [Effect] part represents the expected effect of the action in all situations that match the [Condition] part of the classifier. A set of classifiers constitutes a *model of transitions*, as it is called in the Reinforcement Learning (RL) literature Thus, ALCSs are an instance of model-based RL architectures—a category of systems whose prototype is the DYNA architecture [3]. As a result, ALCSs can be seen as combining two crucial properties of RL systems: Similar to the DYNA architectures, they learn a model of transitions, which endows them with anticipation and planning capabilities and can speed up the learning process. Similar to classical LCSs, they benefit from generalization mechanisms, which enable them to build much more compact models than tabular DYNA architectures [2,4].

On the other hand, in the RL literature, the Factored Markov Decision Processes (FMDPs) framework was introduced to represent large and structured MDPs compactly [5]. In this approach, a state is implicitly described by an assignment of values to some set of state variables—a representation that shares strong similarities with the one used in LCSs where the variables are termed "attributes". But the structure of the model of transitions is assumed to be known in FMDPs, which stands in contrast with the ALCS framework where this structure is learned from experience.

SDYNA [6,7] is a family of systems that perform RL in the FMDP framework where the structure of the model of transitions is learned from experience—an approach that we call Factored Reinforcement Learning (FRL). Thus, like ALCSs, FRL systems are model-based RL systems endowed with a generalization capability.

In this contribution, we examine the conceptual similarities and differences between two ALCSs named MACS and XACS on the one hand, and one instance of SDYNA named SPITI on the other hand. Then we perform an empirical comparison between XACS and SPITI based on two benchmark problems, namely MAZE6 and BLOCKS WORLD. The comparison reveals a conceptual problem in the structured dynamic programming algorithm of SPITI, SVI, from which XACS does not suffer. As a consequence, we discuss the possibility of improving FRL systems based on XACS mechanisms.

The paper is organized as follows. In the next section, we give some background about LCSs, ALCSs, FMDPs, FRL and, in particular, SPITI. Then in Section 3, we highlight conceptual similarities and differences between SPITI, MACS and XACS. In Section 4, we present the experimental study. This comparison shows that XACS outperforms SPITI when the representation used to describe states of the problem can give rise to impossible combinations of values, which is discussed in Section 5 before concluding.

2 Background

2.1 Learning Classifier Systems

Learning Classifier Systems (LCSs) [8] were invented by Holland [9] in order to model the emergence of cognition based on adaptive mechanisms. In LCSs, knowledge is represented by a set of rules called *population of classifiers*, which is evolved by adaptive, usually evolutionary learning mechanisms. In Holland's original work, the cognitive part of the system was implemented by a list of internal messages that related the perception of an agent to its actions through an eventually complex message passing process.

Wilson published two radically simplified versions of the initial LCS architecture, named ZCS [10] and XCS [11], in which the list of internal messages was removed. These (now standard) LCSs use condition-action classifiers and combine RL methods with Genetic Algorithms (GAs) to learn a compact rule sets.

The [Condition] part of classifiers is a list of tests. There are as many tests as attributes in the problem description, each test being applied to a specific attribute. In the most common case where the test specifies a value that an attribute must take for the [Condition] to match, the test is represented just by this value. There exists a particular test, denoted "#" and called "don't care", which means that the [Condition] part of the classifier will match whatever the value of the corresponding attribute. At a more global level, the [Condition] matches if all its tests hold in the current situation. In the case of matching, the classifier may be used to determine current behavior.

2.2 Anticipatory Learning Classifier Systems

Riolo [12] was the first to publish an explicitly anticipatory LCS. His system, CFSC2, was directly inspired by the original LCS architecture of Holland [13] with internal messages.

The first ALCS designed after Wilson's simplifications of the original LCS architectures [10] was ACS [14,15]. Central to ACS, the ALP (*Anticipatory Learning Process*) algorithm is the formal counterpart of Hoffmann's psychological theory of *Anticipatory Behavioral Control* [16]. ACS was later extended by Butz to become ACS2 [17,18] and finally XACS [19]. In parallel, Gérard proposed YACS [20] and MACS [21].

The key difference between LCSs and ALCSs lies in the presence of an [Effect] part in the latter systems. In ACS, ACS2 and YACS, the [Effect] part of each classifier tells which attributes do change and which do not given a certain action is executed in a given situation. To represent this, the [Effect] part can contain a "=" symbol, which means that the corresponding attribute does not change. For instance, classifier `[#0#1] [0] [=10=]` predicts that situation `[1031]` changes into situation `[1101]` given action `[0]` is executed, while situation `[2011]` is predicted to change into `[2101]`. By contrast, MACS uses in the [Effect] part a "?" symbol, which denotes that the classifier cannot predict the value of the considered attribute. The addition of this new symbol results in the capacity to predict the value of each attribute separately at the next time step.

2.3 Factored Markov Decision Processes

The FMDP framework was invented independently from research on LCSs, but it is based on an equivalent formalism. Indeed, an FMDP is described by a set of state variables $S = \{X_i...., X_n\}$, where each X_i takes value in a finite domain $Dom(X_i)$. A state $s \in S$ assigns a value $x_i \in Dom(X_i)$ to each state variable X_i. These variables are the formal counterpart of attributes in the LCS framework.

FMDPs utilize dependencies between variables, defined using Dynamic Bayesian Networks (DBNs) [22], to compactly represent the transition and reward functions of structured MDPs.

The model of the transition of the FMDP is defined by a separate DBN model $T_a = \langle G_a, \{P^a_{X_1}, \ldots, P^a_{X_n}\}\rangle$ for each action a. G_a is a two-layer directed acyclic graph whose nodes are $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$ with X_i a variable at time t and X'_i the same variable at time $t+1$. The parents of X'_i are denoted $\text{Parents}_a(X'_i)$ with $\text{Parents}_a(X'_i) \subseteq X$. The transition model T_a is quantified by *Conditional Probability Distributions* (CPDs), denoted $P^a_{X_i}(X'_i|\text{Parents}_a(X'_i))$, associated to each node $X'_i \in G_a$. In practice, these CPDs can be represented as tables, as rules, as a set of decision trees, or as decision diagrams. In each case, the representation gives the probability distribution of each X'_i given the values of $\text{Parents}_a(X'_i)$. In the case of rules or tables, the generalization property comes from the fact that only the variables belonging to $\text{Parents}_a(X'_i)$ are used to represent the distribution over X'_i. This corresponds to using a "#" for the attributes that correspond to all other variables in the LCS representation.

Given this representation of the transition function and a similar compact representation of the reward function, different dynamic programming algorithms such as SVI and SPI for trees [23] and SPUDD for decision diagrams [24] were shown to converge to the optimal policy [23] while using a representation that is exponentially smaller than the tabular one.

2.4 Factored Reinforcement Learning and SPITI

In FMDPs, the transition function expressed as a set of CPDs is considered known. But for most complex problems, designing these probability distributions by hand is difficult, if not impossible. And to represent them compactly makes things even more difficult.

An alternative consists in learning from experience a model of the transition function under a compact form. If learning the model and dynamic programming backups are performed simultaneously, then this approach is the structured counterpart of *indirect* RL systems, whose prototype is the DYNA architecture.

This insight led to the design of SDYNA as a structured version of the DYNA architecture where the model of transitions and of the reward are learned from experience under a compact form [7]. SPITI is a particular instance of SDYNA. It uses an incremental version of SVI to perform dynamic programming and learns the model of transitions in the form of a collection of decision trees using the Incremental Tree Induction (ITI) algorithm [25].

3 Systems and Comparisons

3.1 Comparing SPITI with MACS

Both MACS and SPITI call upon a model-based RL process and are endowed with a generalization property that makes them able to address large MDPs without prior knowledge of the structure. Furthermore, their representations of the model of the transitions have a similar structure. Indeed, consider an agent in a grid world that perceives whether the eight surrounding cells (starting North and coding clockwise) contain a wall or not (see Figure 1). The formalism in ACS, ACS2, XACS and YACS is able to represent regularities such as *"when the agent perceives a wall to the north, whatever it perceives in any other direction, going north does not produce any sensory change"*, which may be represented by the following classifier if the first attribute corresponds to the value of the North sensor: `[1#######] [North] [========]`. By contrast, MACS can represent regularities between different attributes with a classifier such as `[1########] [Left] [??1??????]`, stating *"when the agent perceives a wall to the north, and turns left, it will perceive a wall on its right"*.

Thus, on the one hand, MACS can represent additional regularities since it can detect regularities between different attributes. However, on the other hand, it only predicts one attribute at a time, whereas the predictions of other ALCSs can be more compact.

Experimental results on model compactness and convergence speed of MACS in grid worlds have shown that it builds a slightly more compact model than YACS, which itself was building models four times more compact than an early version of ACS [21]. Furthermore, MACS was building this model three times faster than YACS, and nine times faster than the early version of ACS counting the number of iterations.

Interestingly, its unique representation of the [Effect] part makes MACS more similar to SPITI than any other ALCS. Indeed, in MACS, the value of each attribute is anticipated separately for each action as a function of a [Condition] part containing variables defining the previous state of the model whereas in SPITI the value of each state variable is anticipated separately for each action as a function of a tree representing the possible combinations of variables defining all previous states of the model. Thus, one classifier in MACS is similar to one branch in the decision tree in the model of transitions of SPITI.

Moreover, MACS is the only ALCS that does not call upon a GA. Instead, to learn the model of transitions it relies on the combination of generalization and specialization heuristics that collaborate to converge towards a compact and accurate model of transitions. This mechanism can be compared more easily with the ITI algorithm used in SPITI, which relies on the χ^2 information metric to grow a decision tree incrementally.

However, beyond these similarities, MACS and SPITI differ in several points. First, MACS represents a deterministic transition model whereas SPITI models a stochastic process through a distribution of probabilities of transition. Secondly, as stated above, building a compact model of the transition function in MACS

relies on a complex combination of heuristics whereas SPITI calls upon the well established ITI algorithm. Thirdly, the model of transitions in MACS is represented as a set of classifiers and the value function is tabular, whereas SPITI implements the model of transitions, the value function and the policy as decision trees. This results in faster algorithmic information access. Finally, and most importantly, in MACS the dynamic programming component of model-based RL is applied to a tabular representation of states, whereas SPITI calls upon SVI to perform this computation compactly—with guarantees of convergence to optimality as far as the model of transitions is perfectly accurate. This computation is very efficient in practice.

All the differences above speak in favor of SPITI that seems mathematically better grounded than MACS and benefits from efficient algorithms. Thus an empirical comparison seems to be pointless. As a matter of fact, we did not perform any experiments comparing SPITI against MACS, since SPITI was shown to perform well on problems that are out of reach of MACS [6,7].

Among these differences, the most crucial one is the fact that MACS does not generalize the models of the reward and the value functions over states. Instead, these models are represented by a table giving a value for each encountered state, which prevents its usage for very large state space problems. Although it shares less similarities with SPITI, XACS is another ALCS that does not suffer from this crucial problem. And, quite interestingly, the experimental comparison that we perform after presenting XACS below reveals that it is endowed with a key property that makes it more efficient than SPITI in the context of large problems where a lot of combinations of state variable values cannot occur.

3.2 Presentation of XACS

The XACS system was developed to overcome the deficiency of not generalizing the value function estimates in MACS [19]. XACS combines two LCSs—the generalizing state transition learner ACS2 [18] and the generalizing function learner XCS, which learns generalized value function estimates in XACS. It was shown that XACS can be robustly applied to blocks world problems, in which previous overgeneralization issues in ACS2 were overcome [19].

Essentially, ACS2 learns a generalized representation of the encountered state-transition function of a problem. It has been shown to reliably learn in various discrete problem domains, being able to ignore irrelevant perceptual attributes, handling noisy inputs, or stochastic state transitions. Knowledge is represented in the aforementioned [Condition], [Action] → [Effect] rules. The rules are learned by a combination of a heuristic, which specializes the rule structures, and a genetic rule generalization mechanism.

The XCS system may be the most well-understood and used LCS to-date. It has been shown to be efficiently applicable in Boolean function problems, real-valued function problems, reinforcement learning problems, and mixed domains including datamining classification [11,26]. XCS learns based on a combination of gradient-based value approximation and genetic algorithm-based rule structure

learning. In combination with ACS2, XCS learns a generalized representation of the state-value function of the encountered reinforcement learning problem. In this case, value approximations are updated similar to the DYNA architecture [19,3].

During learning, XCS and ACS2 create their initial rules by means of a *covering* mechanism, which creates rules with matching conditions given no rules currently match the perceived problem state. For compaction purposes, both systems represent redundant identical rules in one *macro-classifier* rule [11]. The rule structuring mechanisms of either system basically assure that the whole perceived problem space is covered and rules that cover unsampled problem subspaces are forgotten (deleted) over time. Further details on the involved mechanisms as well as theoretic learning bounds can be found in the literature [19,26,27].

During goal-directed behavior, XACS predicts possible next problem states using the model from its ACS2 component, estimates the values of these anticipated states by means of its XCS component, and finally conducts its behavioral decision based on these estimates.

4 Experimental Study

4.1 Maze6

The maze environments are classical LCS benchmark problems. They are represented by a two-dimensional grid. Each cell can be occupied by an obstacle, denoted as attribute value by the character `'O'`, a food item, denoted by `'F'`, or can be empty, denoted by `'.'`. The animat perceives its immediate surrounding starting with the cell to the north and coding clockwise. Thus, the perceptual space in the maze environment $\mathcal{I}_{maze} \subseteq \{., O, F\}^L$ where $L = 8$, the eight adjacent cells. Figure 1 shows MAZE6, one of such standard mazes. For example, an animat located one position below the food perceives `'FOOO..OO'` whereas an animat located at the lower left corner perceives `'.O.OOOOO'`. The simulated animat possesses eight primitive actions, the movements to the eight adjacent cells (i.e. $\mathcal{A}_{maze} = \{N, NE, E, SE, S, SW, W, NW\}$). If a movement leads to a position that is blocked by an obstacle, the action has no effect. Once the food position is entered, the environment provides a reinforcement of 1000 and one

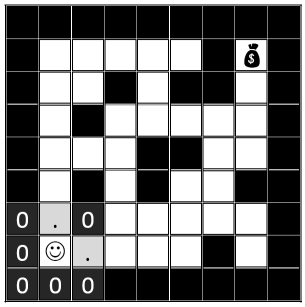


Fig. 1. Maze6

trial ends. In that case, the animat is repositioned to a randomly chosen empty spot in the maze and tries again. Note that, while the state is observed through attributes or random variables, giving rise to an FMDP representation, MAZE6 still obeys the Markov property, that is, the current state perceptions suffice to uniquely identify the current state of the agent, and the knowledge of that state and the action suffice to determine the distribution over next states, by contrast with what would happen in a Partially Observable MDP.

4.2 Blocks World Problem

Our second benchmark is a blocks world scenario introduced in [2]. In this problem, b blocks are distributed over a certain number of stacks s. The agent can manipulate the stacks by the means of a gripper that can either grip or release a block on a certain stack. It perceives the current block distribution coding each stack with b attributes. One additional attribute indicates if the gripper is currently holding a block. Thus, the perceivable situations are a subset of $\mathcal{I} \subset \{*, b\}^{bs+1}$. Additionally, the problem is defined by a particular goal state. We define the goal by putting a particular number y of blocks on the first stack. Figure 2 (right-hand side) shows the goal in the problem with $b = 4, s = 3, y = 3$.

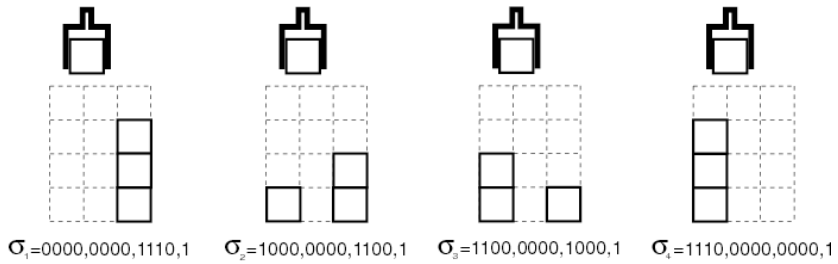
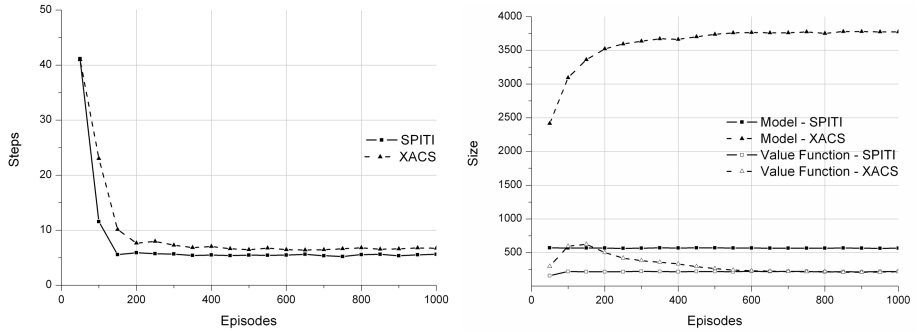


Fig. 2. A blocks world scenario, from a random initial position (left-hand side) to the goal position (right-hand side)

4.3 Experiments

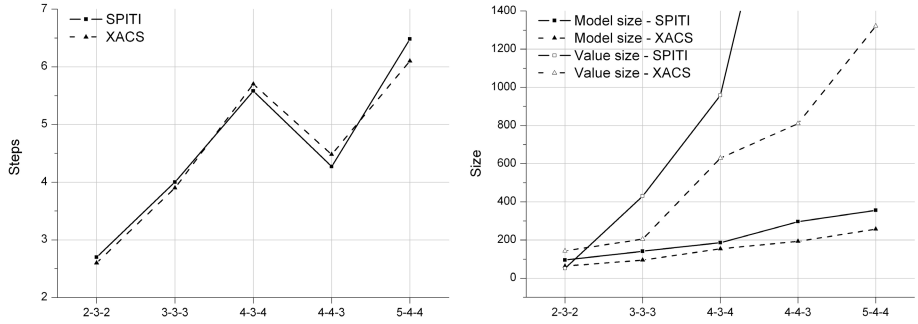
Our experimental protocol is the following. In all runs, we alternate one episode of pure exploration with a random policy and one episode of pure exploitation based on the learned policy. Learning is turned off during exploitation runs. In both benchmark problems, each episode is limited to 50 steps. All results presented below are averaged over 10 runs.

We compare the performance and size of the models in XACS and SPITI. In XACS, the size of the model corresponds to the number of macro-classifiers, for the value function as well as for the model of transitions. In SPITI, it corresponds to the number of branches in the value tree and trees representing the model of transitions.



(a) *(b)*

Fig. 3. *(a)*:Performance in MAZE6 *(b)*:Size of the models



(a) *(b)*

Fig. 4. *(a)*:Performance in BLOCKS WORLD *(b)*:Size of the models

In the case of MAZE6, Figure 3 shows the averaged performance and size of the models from episode to episode.

In the case of BLOCKS WORLD, Figure 4 shows the averaged performance and size of the models in problems with an increasing size, so as to compare the scaling capabilities of both algorithms. In that case, the performance and size for each problem is measured after 200 episodes (alternating 100 exploration episodes and 100 exploitation episodes).

An analysis of Figure 3 shows that, in the case of MAZE6, SPITI slightly outperforms XACS while building a much more compact model of transitions and a model of the value function of similar size after convergence.

By contrast, the analysis of Figure 4 shows that, even if SPITI performs comparably to XACS for small BLOCKS WORLD problems, its model size scales much worse. Thus, XACS can deal with much larger problems than SPITI. The explosion of the size of the model in SPITI also resulted in a much slower computation of the optimal policy.

5 Discussion

The results show that SPITI outperforms XACS on MAZE6 and it performs similar to XACS on small BLOCKS WORLD problems. This suggests that given their clearer mathematical background the basic algorithms in SPITI are intrinsically at least as efficient as the combination of heuristics in XACS. However, the fact that XACS outperforms SPITI on larger BLOCKS WORLD problems and scales much better on these problems reveals a conceptual problem in SPITI that XACS does not suffer from.

The problem is about the representation of "impossible states". In the case of BLOCKS WORLD with the binary representation we used, many arbitrary combinations of attribute values correspond to states that can be represented by both formalisms but that do not occur in practice: all states where a block is lying "in the air" (that is, neither on another block nor on the table) and all states that denote the presence of more or less than b blocks are impossible. The more empty cells in the problem, the more such impossible states. In SPITI, the model of the value function ends with representing explicitly a lot of these impossible states. A closer examination of the algorithms reveals that this undesirable property is inherited from SVI itself, the structured dynamic programming algorithm used in SPITI.

Indeed, in SVI the probabilities of transitions over each variables are computed separately. Thus, the information about the possible or impossible co-occurrences of values of such variables is lost. In the structured Bellman regression algorithm used in SVI, nothing prevents the expression of impossible states in the value function, although these states do not occur in practice. To our knowledge, this fact has never been noticed or made explicit in the literature—seeing also that the benchmark problems used to present structured dynamic programming algorithms are free of such impossible states.

By contrast, XACS benefits from several generalization biases that restrain a possible tendency to represent such impossible states:

- the classifier population is limited in size, resulting in a compactness pressure;
- the covering operator favors the creation of classifier that correspond to actually encountered states rather than impossible ones;
- the genetic-based generalization assures coverage of sufficiently frequently sampled states but also enforces the deletion of rules that cover unsampled subspaces.

In this way, XACS tends to cover the encountered subspace manifold of the full representational space as compactly as possible based on several occurrence and validity signals that are received by means of (random) problem space sampling.

Moreover, due to its interactive specialization and generalization mechanism, XACS identifies the action-dependent state transitions with maximally compact representations. While the specialization mechanism includes seemingly relevant state attributes heuristically, the genetic generalization mechanism deletes over-specializations. While researches might hesitate to utilize the evolutionary-inspired mechanisms used in XACS, comparisons to statistical approaches show

similar functionality and scalability [28,29,30]. Thus, drawing inspiration from XACS suggests to include state occurrence estimates and state relevance estimates, where the latter are based on prediction accuracy estimates, in order to approximate the FMDP model more compactly and efficiently while still sufficiently accurately.

Nevertheless, one must not forget that the model of transitions built by XACS differs from the model in SPITI since XACS calls upon the "=" symbol and predicts several attributes simultaneously whereas MACS uses the "?" symbol and anticipates one attribute at a time, like SPITI. In that respect, on the one hand, a more straightforward SPITI and an ALCS should be the comparison of SPITI with an ideal combination of XACS and MACS– which does not exist so far. On the other hand, trying to figure out whether it is possible to anticipate several attributes at a time within the FRL framework might result in interesting insights.

6 Conclusion

The goal of this contribution was to show that, although ALCSs and FRL systems such as SPITI are conceptually very similar and share interesting properties, they also show some important differences that have major consequences on their algorithmic properties and their performance. By means of an empirical comparison, we have discovered that the structured dynamic programming algorithm, which lies at the heart of one of the main FRL systems, SPITI, suffers from a conceptual problem that prevents it from scaling as efficiently as XACS does—the most efficient ALCS currently available. Future work will need to fix this conceptual problem possibly drawing inspiration from the mechanisms employed in XACS as discussed above.

References

1. Butz, M.V., Sigaud, O., Gérard, P.: Anticipatory behavior: Exploiting knowledge about the future to improve current behavior. In: Butz, M.V., Sigaud, O., Gérard, P. (eds.) Anticipatory Behavior in Adaptive Learning Systems. LNCS, vol. 2684, pp. 1–10. Springer, Heidelberg (2003)
2. Butz, M.V.: Anticipatory Learning Classifier Systems. Kluwer Academic Publishers, Boston (2002)
3. Sutton, R.S.: Planning by incremental dynamic programming. In: Proceedings of the Eighth International Conference on Machine Learning, pp. 353–357. Morgan Kaufmann, San Mateo (1990)
4. Gérard, P., Sigaud, O.: Designing efficient exploration with MACS: Modules and function approximation. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 1882–1893. Springer, Heidelberg (2003)
5. Boutilier, C., Dearden, R., Goldszmidt, M.: Exploiting structure in policy construction. In: Proceedings of the 14th International Joint Conference in Artificial Intelligence, pp. 1104–1111 (1995)

6. Degris, T., Sigaud, O., Wuillemin, P.H.: Chi-square tests driven method for learning the structure of factored MDPs. In: Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence, Massachusetts Institute of Technology, Cambridge, pp. 122–129. AUAI Press (2006)
7. Degris, T., Sigaud, O., Wuillemin, P.H.: Learning the structure of factored markov decision processes in reinforcement learning problems. In: Proceedings of the 23rd International Conference in Machine Learning, pp. 257–264. ACM, Pittsburgh (2006)
8. Sigaud, O., Wilson, S.W.: Learning Classifier Systems: a survey. Journal of Soft Computing 11(11), 1065–1078 (2007)
9. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press, Ann Arbor (1975)
10. Wilson, S.W.: ZCS, a Zeroth level Classifier System. Evolutionary Computation 2(1), 1–18 (1994)
11. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–175 (1995)
12. Riolo, R.L.: Lookahead planning and latent learning in a Classifier System. In: Meyer, J.A., Wilson, S.W. (eds.) From animals to animats: Proceedings of the First International Conference on Simulation of Adaptative Behavior, pp. 316–326. MIT Press, Cambridge (1991)
13. Holland, J.H., Reitman, J.S.: Cognitive Systems based on Adaptive Algorithms. Pattern Directed Inference Systems 7(2), 125–149 (1978)
14. Stolzmann, W.: Anticipatory Classifier Systems. In: Koza, J., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R. (eds.) Proceedings of the 1998 Genetic and Evolutionary Computation Conference, pp. 658–664. Morgan Kaufmann Publishers, Inc., San Francisco (1998)
15. Butz, M.V., Goldberg, D.E., Stolzmann, W.: Introducing a genetic generalization pressure to the Anticipatory Classifier Systems part I: Theoretical approach. In: Proceedings of the 2000 Genetic and Evolutionary Computation Conference (GECCO 2000), pp. 34–41 (2000)
16. Hoffmann, J.: Vorhersage und Erkenntnis [Anticipation and Cognition]. Hogrefe, Göttingen (1993)
17. Butz, M.V.: An Algorithmic Description of ACS2. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS, vol. 2321, pp. 211–229. Springer, Heidelberg (2002)
18. Butz, M.V., Goldberg, D.E., Stolzmann, W.: The Anticipatory Classifier System and Genetic Generalization. Natural Computing 1(4), 427–467 (2002)
19. Butz, M.V., Goldberg, D.E.: Generalized state values in an anticipatory Learning Classifier System. In: Butz, M.V., Sigaud, O., Gérard, P. (eds.) Anticipatory Behavior in Adaptive Learning Systems. LNCS (LNAI), vol. 2684, pp. 282–301. Springer, Heidelberg (2003)
20. Gérard, P., Stolzmann, W., Sigaud, O.: YACS: a new Learning Classifier System with Anticipation. Journal of Soft Computing: Special Issue on Learning Classifier Systems 6(3-4), 216–228 (2002)
21. Gérard, P., Meyer, J.A., Sigaud, O.: Combining latent learning with dynamic programming in MACS. European Journal of Operational Research 160, 614–637 (2005)
22. Dean, T., Kanazawa, K.: A Model for Reasoning about Persistence and Causation. Computational Intelligence 5, 142–150 (1989)

23. Boutilier, C., Dearden, R., Goldszmidt, M.: Stochastic dynamic programming with factored representations. Artificial Intelligence 121(1-2), 10–49 (2000)
24. Hoey, J., St-Aubin, R., Hu, A., Boutilier, C.: SPUDD: Stochastic Planning using Decision Diagrams. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp. 279–288. Morgan Kaufmann, San Francisco (1999)
25. Utgoff, P.E.: Incremental induction of decision trees. Machine Learning 4, 161–186 (1989)
26. Butz, M.V.: Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design. Springer, Heidelberg (2006)
27. Butz, M., Kovacs, T., Lanzi, P.L., Wilson, S.W.: Toward a theory of generalization and learning in XCS. IEEE Transactions on Evolutionary Computation 8(1), 28–46 (2004)
28. Butz, M.V., Lanzi, P.L., Wilson, S.W.: Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction. IEEE Transactions on Evolutionary Computation 12, 355–376 (2008)
29. Potts, D.: Incremental learning of linear model trees. In: Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2004), pp. 663–670 (2004)
30. Schaal, S., Atkeson, C.G.: Constructive incremental learning from only local information. Neural Computation 10, 2047–2084 (1998)