An Experimental Comparison between ATNoSFERES and ACS

Samuel Landau*, Olivier Sigaud*, Sébastien Picault**, and Pierre Gérard*

* Laboratoire d'Informatique de Paris 6
8, rue du Capitaine Scott
75 015 Paris France
{Samuel.Landau,Olivier.Sigaud,Pierre.Gerard}@lip6.fr
http://miriad.lip6.fr/~landau
http://animatlab.lip6.fr/{Sigaud,Gerard}

** Laboratoire d'Informatique Fondamentale de Lille
Cité Scientifique
59 655 Villeneuve d'Ascq Cedex, France
Sebastien.Picault@lifl.fr
http://www.lifl.fr/~picault

Abstract. After two papers comparing ATNoSFERES with XCSM, a Learning Classifier System with internal states, this paper is devoted to a comparison between ATNoSFERES and ACS (an Anticipatory Learning Classifier System). As previously, we focus on the way perceptual aliazing problems encountered in non-Markov environments are solved with both kinds of systems. We shortly present ATNoSFERES, a framework based on an indirect encoding Genetic Algorithm which builds finite-state automata controllers, and we compare it with ACS through two benchmark experiments. The comparison shows that the difference in performance between both system depends on the environment. This raises a discussion of the adequacy of both adaptive mechanisms to particular subclasses of non-Markov problems. Furthermore, since ACS converges much faster than ATNoSFERES, we discuss the need to introduce learning capabilities in our model. As a conclusion, we advocate for the need of more experimental comparisons between different systems in the Learning Classifier System community.

Keywords Evolutionary Algorithms, Perceptual Aliazing, Augmented Transition Networks.

1 Introduction

Most Learning Classifier Systems (LCS) [Hol75] are used to control agents involved in a sensori-motor loop with their environment. Such agents perceive situations through their sensors as vectors of several attributes, each attribute representing a perceived feature of the environment. As pointed out by Lanzi

[Lan00], LCS are adaptive architectures based on *Reinforcement Learning* (RL) techniques [SB98], but endowed with generalization capabilities. Thanks to a LCS, an agent can *learn* the optimal policy – *i.e.* which action to perform in every situation, in order to maximize a reward obtained in the environment. The policy is defined by a set of rules – or classifiers – specifying an action according to some *conditions* concerning the perceived situation.

Standard RL algorithms are generally used in situations where the state of the agent-environment interaction is always known without ambiguity. But in real world environments, it often happens that agents perceive the same situation in several different states, eventually requiring different optimal actions, giving rise to the so called "*perceptual aliazing*" problem. In such a case, the environment is said *non-Markov*, and agents cannot perform optimally if their decision at a given time step only depends on their perceptions at the same time step.

There are several attempts to apply LCSs to non-Markov problems, relying on different approaches to the problem. For instance, in XCSM [Lan98] added explicit internal states to the classical (condition, action) pair of the classifiers used in XCS [Wil95]. From XCS again, [TB00a] proposed in CXCS a rule-chaining mechanism able to build a bridge over ambiguous situations. ACS, an Anticipatory LCS (ALCS), uses a similar rule-chaining mechanism to solve non-Markov problems.

In two recent papers [LPSG02a, LPSG02b], we have presented a new framework, "ATNoSFERES" [LP01], also used to automatically design the behavior of agents and able to cope with non-Markov environments. ATNoSFERES relies on an evolutionary approach instead of classical reinforcement learning techniques, but we have shown in [LPSG02a] that the resulting graph-based representation was semantically very similar to the LCS representation, giving rise to a detailed comparison between both classes of systems. In particular, we have shown that two important advantages of the graph-based representation were its minimality and its readability. As a result, the structure of the controller gives a lot of information about the structure of the problem faced by the system. In these papers, ATNoSFERES was compared with XCSM on the well-known Maze10 environment and then on a new environment called 12-Candlesticks.

In the present paper, we provide a new comparison between ATNoSFERES and another LCS, ACS. We rely on a study from [ML02] to compare the performance of both systems on two distinct environments. Our comparison reveals new features of the interaction of LCSs with non-Markov problems.

In the next section, we summarize the features and properties of the ATNoSFERES model, and we highlight the formal similarity between ATNoSFERES and LCS representations. In section 3, we briefly present the different approaches used in LCSs to cope with non-Markov problems. Then we actually compare ATNoSFERES with ACS in section 4. This new study reveals that some problems found difficult with ACS appear easier with ATNoSFERES and *vice versa*. We discuss this point in section 5. Finally, we draw lessons from the fact that ATNoSFERES converges slower than ACS to conclude that we should include on-line learning mechanisms in our model, and we highlight the need of more

experimental comparisons between classes of Learning Classifier Systems now that the field is getting more mature.

2 The ATNoSFERES model and Learning Classifier Systems

2.1 Graph-based expression of behaviors

The architecture provided by the ATNoSFERES model [LP01, PL01] involves an ATN[1] graph [Woo70] which is basically an oriented, labeled graph with a Start (or initial) node and an End (or final) node (see figure 7). Nodes represent states while edges represent transitions of an automaton.

Like LCSs, ATNoSFERES binds conditions expressed as a set of attributes to actions, and is endowed with the ability to generalize conditions by ignoring some attributes. But in ATNoSFERES, the conditions and actions are used in a graph structure that provides internal states.

The graph describing the behaviors is built from a genotype by adding nodes and edges to a basic structure containing only the *Start* and *End* nodes. The graph-building process was described in [LPSG02a, LPSG02b] and will not be detailed here again. For the self-consistency of the paper, we just have to mention that the process is separated into two steps:

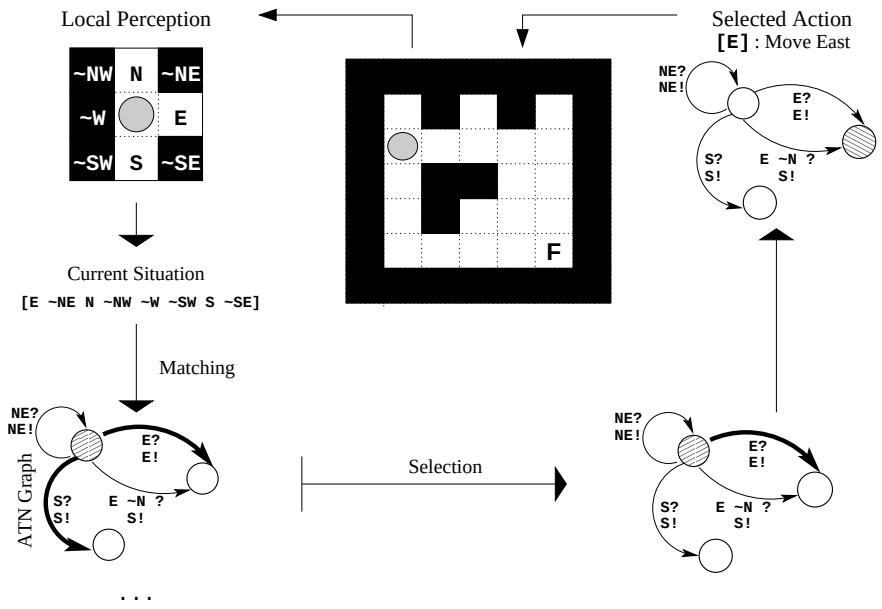
1. The bitstring (genotype) is translated into a sequence of tokens.
2. The tokens are interpreted as instructions of a robust programming language, dedicated to graph building.

Since any sequence of tokens is meaningful, the graph-building language is highly robust to any variations affecting the genotype, thus there is no specific syntactical nor semantical constraint on the genetic operators. In addition, the sequence of tokens is to some extent order-independent and a given graph can be produced from very different genotypes, which guarantees a degeneracy property.

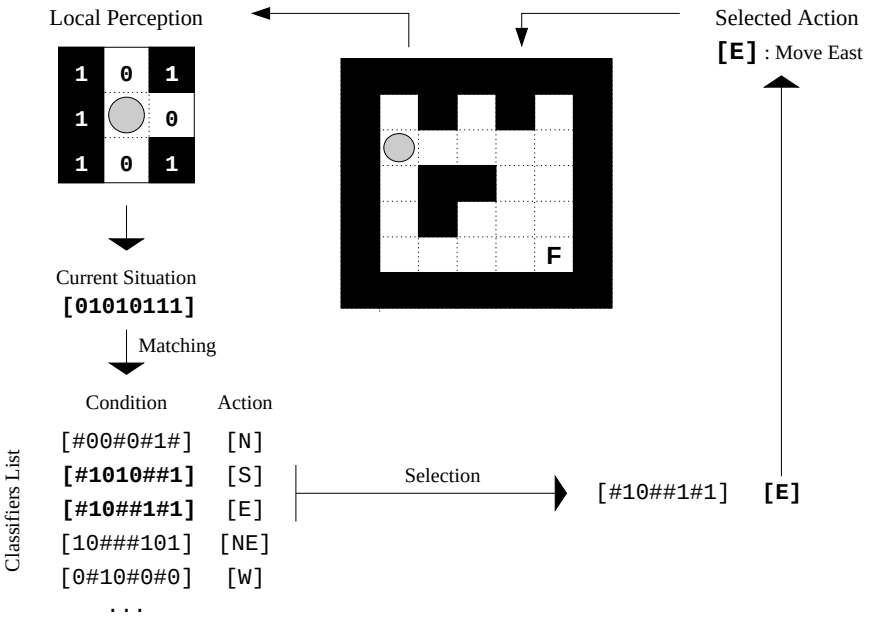
2.2 ATNoSFERES model and Learning Classifier Systems

As explained in more details in [LPSG02a] and illustrated in figure 1, an ATN such as those evolved by ATNoSFERES can be translated into a list of classifiers. The nodes of the ATN play the role of internal states and endow ATNoSFERES with the ability to deal with perceptual aliazing. The edges of the ATN are characterized by several informations which can also be represented in classifiers: the source and destination nodes of the edge correspond to internal states; the conditions associated to the edges correspond to the conditions of the classifiers and the actions associated to the edges correspond to the actions of the classifiers.

[1] ATN stands for "Augmented Transition Networks"



(a) ATNoSFERES



(b) LCS

Fig. 1. The sensori-motor loop with ATNoSFERES and a standard LCS. The agent perceives the presence/absence (resp. 1/0) of blocks in each of the eight surrounding cells and must decide towards which of the eight adjacent cells it should move. In ATNoSFERES, from its current location, the agent perceives `[E ¬NE N ¬NW ¬W ¬SW S ¬SE]` (token `E` is true when the east cell is empty). From the current state (node) of its graph, two edges (in bold) are eligible, since the condition parts of their label match the perceptions. One is selected either deterministically or not, then its action part (move east) is performed and the current state is updated. In a LCS case, the agent perceives `[01010111]` (starting north and rotating clockwise). Within the list of classifiers characterizing it, the LCS first selects those matching the current situation. Then, it selects one of the matching classifiers and the corresponding action is performed.

3 Background: LCSs and non-Markov problems

Dealing with simple `Condition-Action` classifiers does not endow an agent with the ability to behave optimally in perceptually aliazed problems. In such problems, it may happen that the current perception does not provide enough information to always choose the optimal action: as soon as the agent perceives the same situation in different states, it will choose the same action even if this action is inappropriate in some of these states.

For such problems, it is necessary to provide the system with more than just current perceptions. In the general reinforcement learning framework, several kinds of solutions have been tested.

- The first one consists in adding explicit internal states to the perceptions involved in the decisions of the system. This approach was used by Holland in his early LCSs thanks to an internal message list [HR78]. But both [RR88] and [Smi94] reported unsatisfactory performance of Holland's system on non-Markov problems. In the context of more recent LCS research, the explicit internal state solution was adopted by [CR94] in ZCSM and by [LW00] in XCSM and XCSMH.
- The second one, memory window management, is a special case of explicit internal state management where the internal state consists in an immediate memory of the past of length k. Some systems use a fixed size window (see [LM92] for a review) while others use a variable size window (*e.g.* [McC95]). The next solution, rule-chaining, can be seen as an alternative view of the variable size window mechanism.
- The third one consists in chaining the decisions, making one decision depend on the decisions previously taken, so as to use a memory of what was done previously to disambiguate the current situation. Among LCSs, this solution was used in ZCCS [TB00b], CXCS [TB00a] and ACS [Sto99].
- The fourth one consists in splitting a non-Markov problem into several Markov problems, making sure that aliased states are scattered among different sub-problems. This solution has been investigated first by [WS97], and then improved by [SS00]. To our knowledge, no LCS actually uses this solution, despite its very interesting properties.
- The last solution consists in building a finite state automaton corresponding to the structure of the problem, as [MPKK99] or [Han98] do, in a context where the structure of the problem is known in advance. This is the solution chosen in ATNoSFERES, using a Pittsburg style evolutionary algorithm, but in a context where the agents do not know anything about the structure of the problem before starting.

4 Experimental Comparison with ACS

4.1 ACS

In previous papers, we have compared ATNoSFERES with XCSM on two non-Markov problems. In order to go deeper into the comparison between the abilities

of ATNoSFERES and LCSs to cope with the perceptual aliazing problem, we present in this section a comparison with another system, ACS.

The Anticipatory Classifier System has been developed by Stolzmann [Sto98]. It differs from classical Learning Classifier Systems by adding to the perception-action rules an "effect part" that represents a perceptual anticipation of the consequences of the action upon the environment. ACS relies on an Anticipatory Learning Process (ALP) [Sto98] and has been successfully applied to both Markov and non-Markov environments.

The main feature of ACS with respect to XCS-like LCSs relies in the fact that their use of anticipation make it possible to design some efficient heuristics that are believed to make the system converge faster, though no explicit performance comparison has been published yet. Gérard and Sigaud have proposed two ALCSs similar to ACS, namely YACS [GSS01] and MACS [GMS03], that have been shown to be faster than ACS, but are limited to Markov and deterministic environments.

In ACS, in order to deal with non-Markov environments, it was chosen to use a rule-chaining mechanism like in CXCS [TB00a]. In that case, the effect part of a classifier consisting in a behavioral sequence is intended to represent the perceptual consequence of the sequence of actions. As it is the case with CXCS, this feature makes ACS able to deal efficiently with non-Markov environments [Sto99].

In order to build such a behavioral sequence, a new parameter was added to ACS, namely "BS_{max}". BS_{max} represents the maximal length of the behavioral sequences that ACS may build. Its value must be decided before starting any run.

4.2 Experimental setup

We tried to reproduce an experimental setup as close as possible to that used in [Lan98] with the `Maze10` environment and ACS in E1 and E2 environments, taking into account the specificities of our model. This setup has been applied to all the experiments presented in this paper.

Perception/Action abilities and Tokens. The agents used for the experiments are able to perceive the presence/absence of walls or the presence of food in the eight adjacent cells of the grid, these three perceptions being mutually exclusive. They can move in adjacent cells (the move will be effective if the cell is empty or contains food). Thus, the genetic code includes 24 condition tokens, 8 action tokens, 7 stack manipulation tokens and 4 node creation/connection tokens. We used 7 bits encoding to define the tokens ($2^7 = 128$ tokens, which means that some tokens are encoded twice or more).

In [LPSG02b], we demonstrated that the performances of ATNoSFERES could be increased by using a new token, *selfConnect*, endowing our model with the ability to build easily self-connecting edges from a node to itself. This new token has been used in all the experiments presented below.

Course of Experiments. Each experiment involves the following steps:

1. Initialize the population with $N = 300$ agents with random bitstrings.
2. For each generation, build the graph of each agent and evaluate it in the environment.
3. Select the 20 % best individuals of the population and produce new ones by crossing over the parents. The system performs probabilistic mutations (with a 1% rate) and insertions or deletions of codons (with a 0.5% rate) on the bitstring of the offspring.
4. Iterate the process in 2 with the new generation.

Fitness function. Each individual is evaluated by putting it into the environment, starting on a blank cell in the grid, and letting it try to find the food within a limited amount of time (the limit is 20 time steps in all experiments described below). The agent can perceive the food, and it can perform only one action per time step; when this action is incompatible with the environment (*e.g.* go west when the west cell contains an obstacle), it is simply discarded (the agent loses one time step and stays on the same cell).

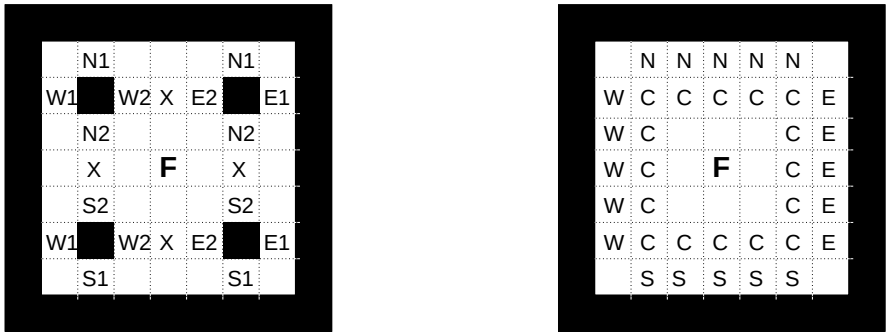
The fitness of the agent for each run is the remaining time if the food has been found within the time limit. Thus, the selection pressure encourages short paths to food. For one generation, each agent is evaluated one time starting on each empty cell, then its total fitness for this generation is the sum of the fitnesses computed for each run. Each agent is reevaluated at each generation in order to average its fitness over generations. This is necessary because of the non-deterministic aspects of the automata.

Indeed, there are several potential sources of non-determinism in our automata. The first one is due to the fact that several arcs might be eligible from the current node in the current situation. In that case, we can either choose one arc randomly, giving rise to a non-deterministic behavior, or assign fixed priorities (by order of creation, for instance) to arcs, so as to keep the automata deterministic. In all the experiments presented here, we have chosen the deterministic stance, after having checked that we obtain better performance with such a choice.

But there are still two sources of non-determinism in our automata. In a situation where no arc is eligible, or when an edge to cross does not carry any action label, one action is chosen randomly. Thus an automaton will be fully deterministic only in the case where one arc can be elected in any encountered situation, and if all such arcs bear an action to perform. This explains the need to average the performance over several runs.

4.3 Experimental environments

The experiments described below take place in two non-Markov environments (E1 and E2, see figure 2) that have been used in [ML02] to study how ACS deals with non-Markov problems. E1 presents 20 aliazed situations (among the

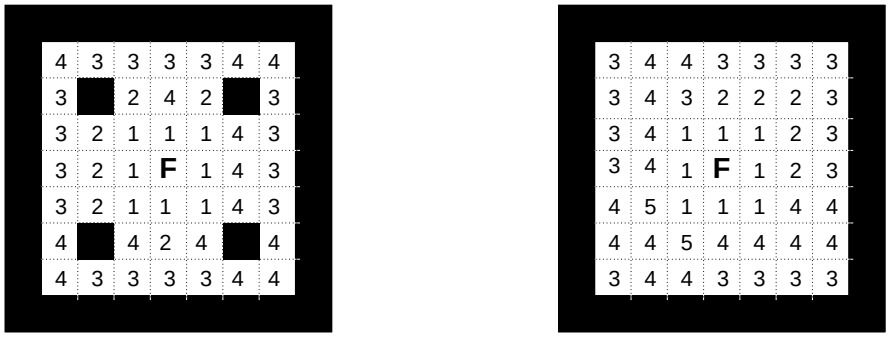


(a) E1 environment (b) E2 environment

Fig. 2. E1 and **E2** environments. **F** (food) is the goal. Other marked cells represent aliazed situations (identical letters imply the same perception).

44 free cells) which are perceived as 9 distinct situations. E2 presents 36 aliazed situations (among 48 free cells), which are perceived as 5 distinct situations.

On figure 3, we show the number of steps an optimal agent among several may need to reach food from each starting cell, given that its perception is limited (an omniscient agent could perform even better).



(a) E1 environment (b) E2 environment

Fig. 3. One optimal policy for E1 (resp. E2), represented by the number of steps needed to reach food from each Start cell. Other equivalent policies can be obtained at least by applying all possible rotations and symmetries to all the numbers given. In E1, the optimal average number of steps to food is 2.8181 steps. In E2, it is 2.9792 steps.

4.4 Comparison with ACS

Before comparing, we have to emphasize a major difference between the way ACS and ATNoSFERES deal with these environments. This difference regards the implicit selection of possible movements. In ACS experiments, as they are described in [ML02, § 4.1 and 4.2], the only movements tested in each free position are transitions towards surrounding free cells (for example, if the cell to the north contains an obstacle, the move to the north is not considered as a possible move, thus it is not tested). This constitutes a kind of prior domain-dependent knowledge about consistent perceptions-actions bindings, which significantly biases the learning process by reducing the number of classifiers to test. In [SG99], we have shown that prohibiting the use of this bias can severely impair some learning algorithms. For instance, McCallum's U-Tree algorithm [McC95] which works well in non-Markov mazes such as those studied here if the agent is prevented from bumping into walls, might grow an infinitely deep tree if it keeps bumping into the same wall in an aliased situation.

In ATNoSFERES, on the contrary, any move token can be used as an action label. When the corresponding movement is impossible, the agent stays where it is and loses a time step (it is penalized only in an indirect way, through the fitness function).

The experiments reported here were carried out on various initial genotype sizes. In E1, the genotypes that have been tested are between 40 and 150 tokens long (with step 10), as in E2. Using these different sizes was necessary because we do not know in advance the minimum size required to produce an efficient automaton.

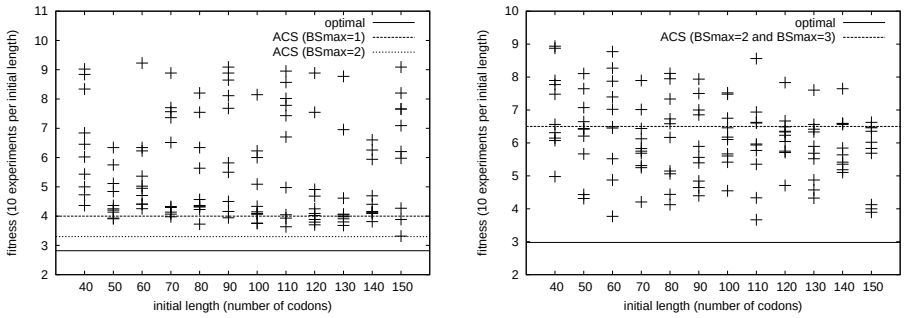
The original population genotype sizes may drift during an evolution, since some genetic operators insert or delete parts of the genotype randomly. Each experiment is stopped after 10,000 generations, and 10 experiments have been performed in each experimental situation.

4.5 Results

Figure 4 gives the respective fitness values obtained by the best automata in E1 and E2 experiments, depending on initial lengths of the genotypes. Each cross in the figures represent the performance of the best automaton obtained after 10,000 generations in one run. Thus there are ten crosses for each initial length. From figure 4 (a), it can be seen that in E1, ATNoSFERES easily reaches the performance of ACS in the case where $BS_{max} = 1$, but hardly reaches the performance of ACS with $BS_{max} = 2$, which is very close to the optimal performance.

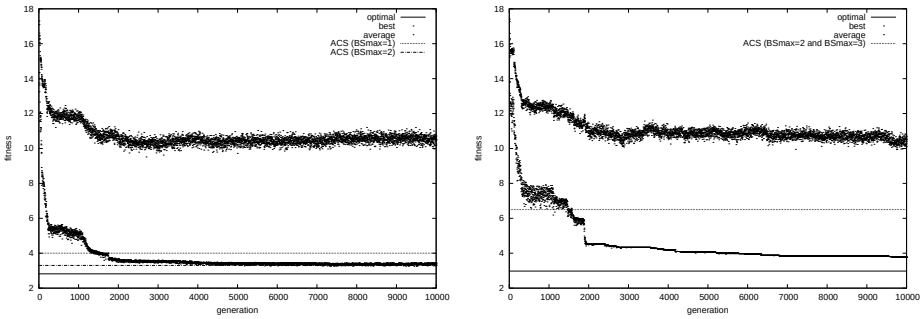
In E2, the performance obtained with ATNoSFERES is significantly better than the one obtained with ACS with $BS_{max} = 2$ and $BS_{max} = 3$. Indeed, ATNoSFERES is about twice closer to the optimum performance.

In order to check whether ATNoSFERES could reach an even higher performance in E1, we took the best run on figure 4 (a) and ran it up to 100,000



(a) E1 environment (b) E2 environment

Fig. 4. Minimal average time to reach food in E1 (resp. E2) experiments with "deterministic" automata as a function of the initial length of the bitstring.



(a) E1 environment (b) E2 environment

Fig. 5. Best fitness evolution in E1 (resp. E2) experiment as a function of generations; the shape and smoothness of the curve are representative for all E1 (resp. E2) evolutions. The thickness of the curves (particularly manifest in E1) is due to the indeterministic behavior of agents. In E2, it seems that the pressure towards deterministic behavior is stronger.

generations. The best performance was slightly improved again, reaching 3.2 (it was 3.3 after 10,000 runs).

Figure 5 gives the evolution of the best fitnesses, respectively in E1 and E2 environments. It appears clearly that gradual improvements occur in both environments.

4.6 Representative solutions

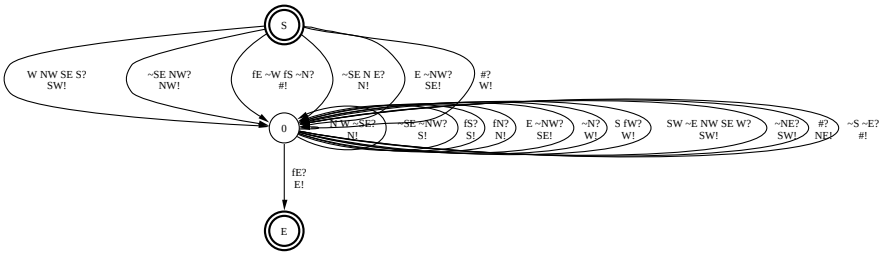


Fig. 6. A representative automaton found with ATNoSFERES in E1 experiment (after 10,000 generations). Its average number of steps to food is about 3.8

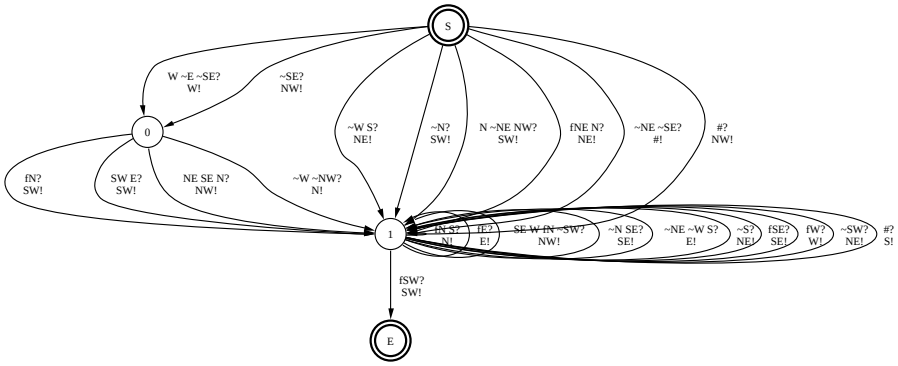


Fig. 7. The best automaton found with ATNoSFERES in E1 experiment (after 10,000 generations). Its average number of steps to food is about 3.3

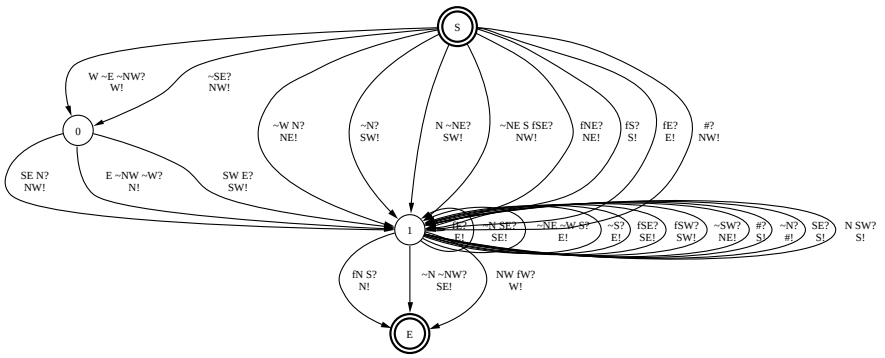
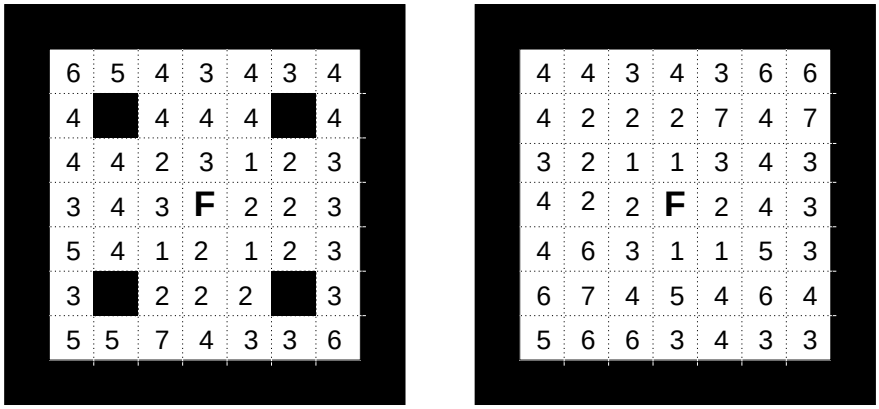


Fig. 8. The best automaton found with ATNoSFERES in E1 experiment (after 100,000 generations). Its average number of steps to food is about 3.2

E1 environment We present on figure 8 the best automaton obtained in E1 experiments after 100,000 generations, on figure 7 the best automaton obtained after 10,000 generations, and on figure 6 a more representative automaton obtained after 10,000 generations. From these figures it is clear that the most common solutions found are nearly reactive. The graph of the more common automata contains a single node (in addition to the Start and End node that always exist in ATNoSFERES graphs), which means that a reactive behavior already performs well in E1. The results show that this kind of behavior is produced in most cases and gets high fitness values, more easily than solutions involving internal states.

However, the automaton depicted on figure 7 shows that adding one node can already improve significantly the global performance.

The main difference between the best automaton obtained after 10,000 generations and the one obtained after 100,000 generations is that the latter contains several additional arcs. In particular, the agent will more often take into account the presence of food (label f on the edges) in its immediate surrounding to reach it immediately.



(a) E1 environment (b) E2 environment

Fig. 9. Best policy found with ATNoSFERES in E1 (resp. E2) in 10,000 generations, represented by the number of steps needed to reach food from each Start cell (see figure 3 for optimal policy).

Indeed, we can see on figure 9 (a) that in several situations where the food is visible the agent needs more than one step to reach it, though a more efficient behavior is obvious. ATNoSFERES has a lot of difficulties in finding these re-

active rules that a reinforcement learning algorithm combining exploration and exploitation would find immediately.

However, even if these additional arcs could improve the performance a bit more, this would not be enough to reach the true optimal performance. A carefully hand-crafted optimal automaton needs much more internal states than the ones shown in this section.

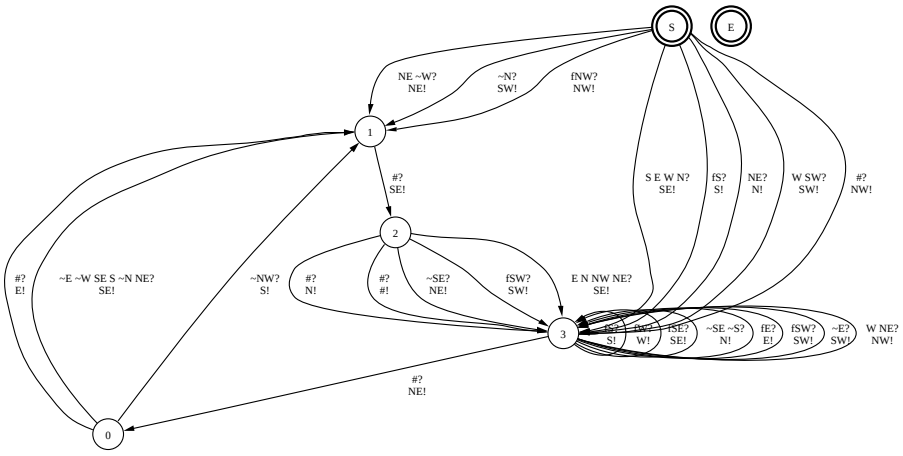


Fig. 10. The best automaton found with ATNoSFERES in E2 experiment. Its average number of steps to food is about 3.8

E2 environment Figure 10 gives the best automaton found in E2 environment. From this figure it is immediately clear that a good automaton in E2 needs more nodes than it is the case in E1. This seems to imply that reactive and nearly reactive behaviors perform much worse in E2 than in E1. This fact, in addition to the fact that ATNoSFERES clearly outperforms ACS on E2 while it is less the case in E1, will be at the heart of the discussion that follows.

5 Discussion

The experimental study presented in the previous section reveals that different subclasses of non-Markov problems should be distinguished more accurately. Indeed, some problems, like E1, are actually non-Markov, but in such a way that reactive behaviors can still perform well on such problems.

In E1, our study has shown that through an evolutionary process, it is easy to gradually grow a set of *ad hoc* rules (which are to some extent independent from each other), even more if the agent is tested from each cell: thus, an agent

can start with a few rules that are efficient for a few cells, and evolve from one generation to another rules that are useful for additional cells. From such a reactive solution, built by the accumulation of small changes, it is unlikely to develop internal states to deal with a few particular cases, since it requires at the same time additional nodes, linked with consistent edges, conditions and actions. We meet again the *structural cost* mentioned in [LPSG02a]: "simple", incremental good solutions are preferred to structurally complex optima.

On the contrary, other problems, like E2, should be said "highly non-Markov", since reactive policies perform very poorly on such problems. In E2, there is no hope that a reactive behavior could lead to the food in a reasonable amount of time, due to the location and the nature of aliazed situations.

Our comparative study has revealed that ACS performs very well on the first subclass of problems and more poorly on the second, while ATNoSFERES performs consistently on both subclasses.

Now we should ask ourselves why this is so. On first thoughts, one might consider that the maximal length of sequences in ACS plays a major role in the phenomenon. One could expect that setting BS_{max} to more than 3 in E2 should fix the problem. A closer examination, however, reveals that this is not so.

In [ML02], the authors show that setting BS_{max} to 3 is enough to let ACS build a completely reliable model of E2, under the form of `(situation, action, next situation)` classifiers. This explains why they did not try $BS_{max} = 4$ or more.

But the performance concern and the model reliability concern are not strictly correlated. Regarding the convergence to stable reward performance, [ML02] emphasize that increasing the maximum length of the behavioral sequence "does not improve the 'steps to food' performances", *i.e.* a "good" behavioral solution can be exploited without having built an exhaustive representation of the environment.

One reason explaining that building longer action sequences would not improve the performance comes from the fact that these sequences specify a blind series of actions to perform without interruption and without checking between its beginning and its end the situation perceived in the environment by the agent. These sequences can improve the performance of the agent when they let it jump over ambiguous situations, but they have two main drawbacks:

- first, they do not help the agent when it is starting from an ambiguous situation, since at the first time step the agent benefits from no memory to help disambiguating its situation;
- second, once a sequence is elected, the agent will at least perform the number of actions specified in the sequence.

Since the number of steps to the food given by the optimal policy in E1 and E2 is generally less than 4, it is very unlikely that letting the agent perform sequences of 4 actions or more will help reaching the optimal performance.

Even worse, if an agent starts from an ambiguous situation and then follows a long sequence of actions, this sequence will delay the time at which the agent can discover its actual location and then follow an optimal path to the food.

Indeed, our experience with ATNoSFERES in small environments like E1 and E2 is that the main issue for the agent consists in discovering as fast as possible where it is from an initially ambiguous situation and then follow the shortest path to the goal. Maybe the situation about the use of sequences would be different in much bigger environments, but we will not treat this issue here.

Finally, we must compare the number of elementary runs necessary to reach a good performance with ACS and ATNoSFERES. In the experiments reported in [ML02], ACS needs about 60,000 steps (resp. 120,000 steps) to build an exhaustive internal model of E1 (resp. E2) given a convenient length of the behavioral sequence used as action part in ACS. With ATNoSFERES, about 1500 generations of 300 individuals are necessary to obtain a performance similar to that of ACS with $BS_{max} = 1$ in E1 and $BS_{max} = 2$ or 3 in E2, which makes about 450,000 runs of 6 to 15 steps on average. Thus it is clear that ATNoSFERES still needs several orders of magnitude more steps than ACS to converge.

This can be easily explained by the fact that ATNoSFERES evolves automata thanks to a blind GA process while ACS relies on a reinforcement learning algorithm which extracts information about the environment from its experiences. From this comparison, it is clear that an area for a major improvement of ATNoSFERES consists in endowing it with reinforcement learning capabilities. This is our immediate agenda for future work.

A source of inspiration in that direction comes from the SAMUEL system [Gre91]. Like ATNoSFERES, SAMUEL is a *Pittsburg* style system based on a single chromosome GA, but it also includes lamarckian operators that endow it with basic learning capabilities. As a result, as claimed by the author, "Samuel represents an integration of the major genetic approaches to machine learning, the Michigan approach and the Pittsburg approach". Most of the operators used in SAMUEL can be transposed in ATNoSFERES, the main difference being that ATNoSFERES does not provide a high level symbolic representation and that SAMUEL does not include any mechanism to solve perceptual aliasing problems.

6 Conclusion and Future Work

In this paper, we have applied ATNoSFERES to non-Markov environments that have been investigated with ACS. Our experiments confirm that ATNoSFERES encounters more difficulties in producing an optimal behavior in some environments where reactive solutions are highly valuable than in environments that are more difficult for ACS.

Such a result suggests that the difficulties of different non-Markov problems with different hidden-state structure such as E1 and E2 should be distinguished in more details than is usually done. Along that line, we believe that, thanks to the information ATNoSFERES provides on the structure of different problems, it can be seen as a tool that may help understanding which kind of system will perform best in which kind of environment and why.

Finally, we would like to highlight the fact that the comparative studies we provided with ATNoSFERES both in this paper and in [LPSG02a] and

[LPSG02b] should be generalized in the LCS community. Previously, we have compared ATNoSFERES with XCSM on some environments qualitatively, without comparing both systems performances. Here we have compared ATNoSFERES with ACS quantitatively on other environments, relying on the experiments presented on the available literature. Since XCSM and ACS have not been tested on the same environments, a precise comparison of their respective performance has never been published yet. A lot of work deserves to be done to provide more global comparisons between several systems and classes of systems. We strongly believe that such comparisons would greatly enhance the understanding of the current state of the art in the LCS research community.

References

[CR94] D. Cliff and S. Ross. Adding memory to ZCS. *Adaptive Behavior*, 3(2):101–150, 1994.

[DM99] Alexis Drogoul and Jean-Arcady Meyer, editors. *Intelligence artificielle située – cerveau, corps et environnement*, Paris, 1999. Hermès. (In French).

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

[Gre91] John J. Grefenstette. Lamarckian learning in multi-agent environments. In Rick Belew and Lashon Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 303–310, San Mateo, CA, 1991. Morgan Kaufmann.

[GSS01] P. Gérard, W. Stolzmann, and O. Sigaud. YACS: a new Learning Classifier System with Anticipation. *Journal of Soft Computing*, 2001.

[Han98] E. A. Hansen. *Finite Memory Control of Partially Observable Systems.* PhD thesis, University of Massachusetts, Amherst, MA, 1998.

[Hol75] John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* University of Michigan Press, Ann Arbor, MI, 1975.

[HR78] J. H. Holland and J. S. Reitman. Cognitive Systems based on adaptive algorithms. *Pattern Directed Inference Systems*, 7(2):125–149, 1978.

[KBC+98] John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors. *Proceedings of the Third Annual Conference on Genetic Programming*, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

[Lan98] Pier-Luca Lanzi. An Analysis of the Memory Mechanism of XCSM. In Koza et al. [KBC+98].

[Lan00] Pier-Luca Lanzi. Learning Classifier Systems from a Reinforcement Learning Perspective. Technical report, Dip. di Elettronica e Informazione, Politecnico di Milano, 2000.

[LM92] L.-J. Lin and T. M. Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, School of Computer Science, 1992.

[LP01] Samuel Landau and Sébastien Picault. ATNoSFERES: a Model for Evolutive Agent Behaviors. In *Proceedings of the AISB'01 Symposium on Adaptive Agents and Multi-Agent Systems*, 2001.

[LPSG02a] Samuel Landau, Sébastien Picault, Oliver Sigaud, and Pierre Gérard. A comparison between ATNoSFERES and XCSM. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 926–933, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[LPSG02b] Samuel Landau, Sébastien Picault, Olivier Sigaud, and Pierre Gérard. Further Comparison between ATNoSFERES and XCSM. In Stolzmann et al. [SLW02].

[LSW00] P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, editors. *Learning Classifier Systems. From Foundations to Applications*, volume 1813 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 2000.

[LW00] Pier-Luca Lanzi and Stewart W. Wilson. Toward optimal classifier system performance in non-markov environments. *Evolutionary Computation*, 8(4):393–418, 2000.

[McC95] R. A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Rochester, NY, 1995.

[ML02] Marc Métivier and Claude Lattaud. Anticipatory Classifier System using Behavioral Sequences in Non-Markov Environments. In Stolzmann et al. [SLW02], pages 143–163.

[MPKK99] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *Fifteenth Conference on Uncertainty in Artificial Intelligence. AAAI*, pages 427–436, 1999.

[PL01] Sébastien Picault and Samuel Landau. Ethogenetics and the Evolutionary Design of Agent Behaviors. In Nagib Callaos, Susanna Esquivel, and Jamika Burge, editors, *Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI'01)*, volume III, pages 528–533, 2001.

[RR88] G. G. Robertson and R. L. Riolo. A tale of two classifier systems. *Machine Learning*, 3:139–159, 1988.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, an introduction*. MIT Press, Cambridge, MA, 1998.

[SG99] O. Sigaud and P. Gérard. Contribution au problème de la sélection de l'action en environnement partiellement observable. In Drogoul and Meyer [DM99], pages 129–146. (In French).

[SLW02] Wolfgang Stolzmann, Pier-Luca Lanzi, and Stewart W. Wilson, editors. *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS'02)*, LNAI, Granada, september 2002. Springer-Verlag.

[Smi94] R. E. Smith. Memory exploitation in learning classifier systems. *Evolutionary Computation*, 2(3):199–220, 1994.

[SS00] R. Sun and C. Sessions. Multi-agent reinforcement learning with bidding for segmenting action sequences. In J.-A. Meyer, S. W. Wilson, A. Berthoz, H. Roitblat, and D. Floreano, editors, *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*, pages 317–324, Paris, 2000. MIT Press.

[Sto98] W. Stolzmann. Anticipatory Classifier Systems. In Koza et al. [KBC+98], pages 658–664.

[Sto99] W. Stolzmann. Latent Learning in Khepera Robots with Anticipatory Classifier Systems. In Wu [Wu99], pages 290–297.

[TB00a] A. Tomlinson and L. Bull. CXCS. In P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems: from Foundations to Applications*, pages 194–208. Springer Verlag, Heidelberg, 2000.

[TB00b] Andy Tomlinson and Larry Bull. A zeroth level corporate classifier system. In Lanzi et al. [LSW00], pages 306–313.

[Wil95] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[Woo70] William A. Woods. Transition Networks Grammars for Natural Language Analysis. *Communications of the Association for the Computational Machinery*, 13(10):591–606, 1970.

[WS97] M. Wiering and J. Schmidhüber. HQ-Learning. *Adaptive Behavior*, 6(2):219–246, 1997.

[Wu99] A. S. Wu, editor. *Proceedings of the 1999 Genetic and Evolutionary Computation Conference (GECCO'99)*, 1999.