ATNoSFERES revisited

Samuel Landau, Olivier Sigaud, Marc Schoenauer

ATNoSFERES revisited

Samuel Landau
Équipe TAO - INRIA Futurs
LRI - Bât. 490
Université de Paris-Sud
91405 Orsay, France
Samuel.Landau@lri.fr

Olivier Sigaud
LIP6
Université Pierre et Marie Curie
Paris, France
Olivier.Sigaud@lip6.fr

Marc Schoenauer
Équipe TAO - INRIA Futurs
LRI - Bât. 490
Université de Paris-Sud
91405 Orsay, France
Marc.Schoenauer@lri.fr

ABSTRACT

ATNoSFERES is a Pittsburgh style Learning Classifier System (LCS) in which the rules are represented as edges of an Augmented Transition Network. Genotypes are strings of tokens of a stack-based language, whose execution builds the labeled graph. The original ATNoSFERES, using a bitstring to represent the language tokens, has been favorably compared in previous work to several Michigan style LCSs architectures in the context of Non Markov problems. Several modifications of ATNoSFERES are proposed here: the most important one conceptually being a representational change: each token is now represented by an integer, hence the genotype is a string of integers; several other modifications of the underlying grammar language are also proposed. The resulting ATNoSFERES-II is validated on several standard animat Non Markov problems, on which it outperforms all previously published results in the LCS literature. The reasons for these improvement are carefully analyzed, and some assumptions are proposed on the underlying mechanisms in order to explain these good results.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Experimentation

Keywords

Learning Classifier Systems, ATNoSFERES, Partially Observable Markov Decision Processes

1. INTRODUCTION

The Pittsburgh versus Michigan debate has been present in the Learning Classifier Systems (LCSs) community since the very beginning of this research area (see [14] for a presentation). On the one hand, most recent works in the LCS domain have chosen the Michigan approach [7]. The ATNoSFERES architecture proposed in [3, 5], on the other hand, is a Pittsburgh style architecture dedicated to the resolution of non Markov problems. It has been compared to several LCSs, in particular in [4], where the authors concluded a systematic comparison by the claim that the ATNoSFERES approach was more robust and more general than its Michigan style opponents, but that it was also several orders of magnitude slower.

This paper investigates the effect of several modifications to the original ATNoSFERES approach on its global efficiency. Instead of being represented by a bitstring, the genotype, that encodes a series of tokens of the chosen stack-base graph-building language, is now a string of integers. This new representation greatly impacts on the mutation, that can now be naturally made uniform on the set of tokens. Moreover, all *stack* tokens of the underlying language can now operate on specific data on the stack. Finally, the random default behavior when no edge is eligible in the current node of the automaton is suppressed, and the contradictions in the conditions of all edges of the control graph are filtered out. Because all those changes tend to make ATNoSFERES representation and behavior closer to the actual semantic of the underlying search space – that of the ATNs – it is hoped that the resulting algorithm, termed ATNoSFERES-II, achieves at least as good results as ATNoSFERES, but much faster.

The paper is organized as follows: the next section gives a short overview of the previously published versions of ATNoSFERES. Section 3 details the modifications introduced in this paper. Section 4 indicates the experimental conditions used to test ATNoSFERES-II on classical Non Markov animat problems. Section 5 presents the results obtained by ATNoSFERES-II, compares them to the state-of-the-art LCS algorithms and produces statistical significance tests of the improvements. Finally, all those results are discussed in section 6, and some conclusions are draw on the impact of the proposed modifications.

GECCO'05, June 25–29, 2005, Washington, DC, USA.

2. OVERVIEW OF ATNoSFERES

2.1 The bitstring representation

The ATNoSFERES model [2] is designed to generate the control architecture of agents thanks to an evolutionary algorithm. The control architecture itself is represented as an

ATN [1] graph where nodes represent states and edges represent transitions of an automaton. The graph describing the behaviors is built by interpreting a genotype as a program in a stack-based language [1] that proceeds by adding nodes and edges to a basic structure initially containing only the *Start* and *End* nodes. This genotype is a bitstring in the original ATNoSFERES model.

The graph-building process operates in two steps. First, during *translation*, the genotype is translated into a sequence of tokens (see table 1). Second, during *interpretation*, the token interpreter is fed with the token stream produced by the translator. The tokens are interpreted one by one as instructions of a robust programming language dedicated to graph building. The interpretation of each successive token operates on a stack in which parts of the future graph are stored. The construction of the graph takes place during this interpretation process, by creating nodes and connections between nodes. When all tokens have been interpreted, the nodes (each one carrying connections to other nodes) are popped from the stack and the graph is ready to use.

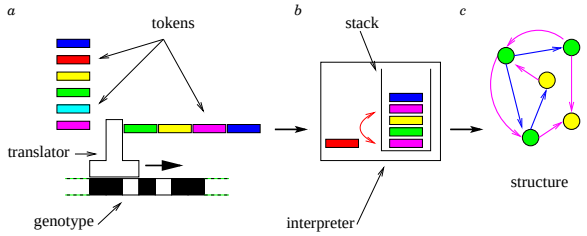

Figure 1: Principles of the genetic expression used to produce the behavioral graph from the bitstring genotype. The string is first decoded into tokens (*a*), which are interpreted in a second step as instructions (*b*) to create nodes, edges, and labels. Finally, when all tokens have been interpreted, the unused conditions and actions remaining in the stack are added into the structure and the structure is popped from the stack (*c*).

There are three kinds of tokens: *action and condition* tokens, such as *SW!* or *foodNE?*, that are action and condition labels for the edges of the ATNs, specific to the actions (movements) and perceptions (what is in a nearby cell) of the agent in the maze environments; *graph structure* tokens, such as *node* or *connect*, that are instructions to push nodes and connect them with edges, using the labels in the stack; and finally *stack* tokens, such as *swap all* or *roll node*, that manipulate the stack either as a whole (the *all* family of stack tokens) or by acting only on one of the two kinds of data present in the stack (the *node* and the *label* families of stack tokens).

2.2 The evolutionary framework

A generation of ATNoSFERES classically starts with the selection of the parents that will be selected for reproduction, then applies variation operators (crossover and mutation) to these lucky parents.

The selection is a rank-based truncated selection that proceeds as follows: First, the n best individuals of the population (of size P) are retained (deterministic truncation), and sorted by descending fitness. They will be used to generate $P-n$ offspring that will complete the population. Then, in order to generate each pair of offspring, two parents are selected from those n best, based on a rank-based exponential selection: if the best parent is associated with a bias b, the i^{th} best parent is associated with bias $b.c^i$, for some $c \in]0,1[$. Thus, of course b, c and n must satisfy $\sum_{i=0}^{n-1} b.c^i = 1$. In all experiments described throughout this paper, $P = 300$, $n = 60$, and c was chosen so that the selection pressure after truncation (bias from the best to the worst of the n parents) is 2 (i.e. $c = \sqrt[60]{0.5}$).

After the selection described above, the genotype of two offspring is produced by a 2-points crossover between the two genotypes, where crossover points must be at the border between two tokens.

Additionally, in the original ATNoSFERES approach, two different mutation strategies were used: classical bit-flip mutation, and random insertions or deletions of one codon (i.e. one token of the language at hand). However, using add/delete mutations did not improve the performances of ATNoSFERES, and only the bit-flip mutation was ever used. This modifies the sequence of tokens produced by translation, so that the complexity of the graph itself may change. Nodes or edges can hence be added or removed by the evolutionary process, as can condition/action labels on the edges.

2.3 The evaluation function

The fitness of each genotype is assessed by first building the ATN, as described above, then by putting this ATN as the controller of an agent and evaluating the behavior of the agent in a (Non Markov) environment. There are three sources of non-determinism in the use of the ATN, therefore, each parent is re-evaluated together with the offspring, and the fitness is averaged over successive evaluations (by calculating the mean fitness over all evaluations).

To evaluate an agent, the ATN graphs are used as follows for at most a fixed number of time steps:

- At the beginning (when the agent is initialized), the agent is at the *Start* node (S).
- At each time step, the agent crosses an edge:
  1. It computes the set of eligible edges among those starting from the current node. An edge is eligible when either it has no condition label or all the conditions on its label are simultaneously true.
  2. An edge is chosen in this set. The first versions of ATNoSFERES were selecting one edge randomly (this is the first source of non-determinism), but it was found that a deterministic choice (e.g. choose the first edge in the list of eligible edges) held better results. If the set of eligible edges is empty, then an action is chosen randomly over all possible actions, and the current node remains unchanged (this is the second source of non-determinism).
  3. The actions on the label of the elected edge are sequentially performed by the system. Assuming that only one action can be performed at a time, only the last action is actually performed. When the action part of the label is empty, an action is chosen randomly. This is the third and last source of non-determinism. However, in all experiments described in this paper, the action

[1] Augmented Transition Network [15]

000000	*swap all—node*	000001	*swap all—node*	000010	*swap all—label*	000011	*swap all—label*
000100	*dup label*	000101	*dup label*	000110	*dup node*	000111	*dup node*
001000	*del label*	001001	*del label*	001010	*del node*	001011	*del node*
001100	*roll all—node*	001101	*roll all—label*	001110	*unroll all—node*	001111	*unroll all—label*
010000	*node*	010001	*node*	010010	*node*	010011	*node*
010100	*connect*	010101	*connect*	010110	*connect*	010111	*connect*
011000	*connect self*	011001	*connect self*	011010	*connect self*	011011	*connect start*
011100	*connect start*	011101	*connect start*	011110	*connect end*	011111	*connect end*
100000	*goN!*	100001	*goS!*	100010	*goW!*	100011	*goE!*
100100	*goNE!*	100101	*goSE!*	100110	*goNW!*	100111	*goSW!*
101000	*emptyN?*	101001	*foodN?*	101010	*treeN?*	101011	*emptyS?*
101100	*foodS?*	101101	*treeS?*	101110	*emptyW?*	101111	*foodW?*
110000	*treeW?*	110001	*emptyE?*	110010	*foodE?*	110011	*treeE?*
110100	*emptyNE?*	110101	*foodNE?*	110110	*treeNE?*	110111	*emptySE?*
111000	*foodSE?*	111001	*treeSE?*	111010	*emptyNW?*	111011	*foodNW?*
111100	*treeNW?*	111101	*emptySW?*	111110	*foodSW?*	111111	*treeSW?*

Table 1: The genetic code used in the experiments, for 6-bit codons. This is the mapping from bitstrings/integers (in binary representation) to tokens. Some codes for the swap, roll and unroll stack tokens have two alternative behaviors, either *all* or *node/label* (see the text, section 3.4). Also note that, due to the ATNoSFERES requirement that there are 2^N tokens for some N, some tokens are represented more than once (e.g. *swap all* is represented 4 times), inducing a slight bias toward those nodes.

part of all edges will contain at most one action, in order to simplify the comparison with LCSs.

4. The target node of the elected edge becomes the new current node.

- The agent stops when it reaches the *End* node (E). This node is a general feature of the model and may never be reached, either if the agent loops infinitely or if the experiment is stopped before.

3. ATNOSFERES-II

This section introduces the modifications that have been brought to the original ATNoSFERES model described in the previous section. In order to be able to assess the effects of each modification independently of one another, all combinations will be made possible, even if the algorithm referred to as ATNoSFERES-II uses them all.

3.1 Integer representation

Instead of coding a genome as a bitstring, a string of integers is now used, where each integer is an index encoding a token (in the range $[0, \#tokens]$). There might be repeated tokens, which is the case in our experiments. First, this simplifies the translation process (the decoding of a bitstring into an integer disappears). However, and because crossover was only allowed at the boundary between tokens in the bitstring genotype of the original ATNoSFERES, the only visible effect of this deep modification is at the mutation level: it is now possible to use the uniform mutation operator that replaces a given token by another one uniformly, i.e. with equal chance for all tokens in the list.

Another possibility could be to define a distance among the tokens based on their semantics (their effect on the structure being built), in order to bias the mutation. Such a bias would be used to smooth mutation strength by having smaller effects on the phenotype change. However, it is clear that the bitflip mutation used in the original ATNoSFERES was in fact equivalent to such a weighted mutation, but the mutation biases were dependent on the order of the tokens in the token list (e.g. by having condition and action tokens at the second half of the list, the first bit was a flag for condition/action tokens). The induced mutation biases were thus completely arbitrary with respect to the problem at hand. Another important difference is that the total number of encoded tokens no longer has to be 2^N for some N (where N is the number of bits used to represent a token).

Note that a similar effect could also have been obtained in the bitstring context by increasing the bitstring length and decoding each token using a *modulo* function, as is theoretically proved in [10].

In the following, and because the only visible effect of this change of representation is the actual change of mutation, the original ATNoSFERES approach will be referred to as *BitFlip* while the ATNoSFERES-II approach will be termed *Uniform*.

3.2 Default node action

As stated in section 2.2, in the original version of ATNoSFERES, when no edge could be elected, an action was chosen randomly – this was referred to as the second source of non-determinism. However, this results in non-deterministic performance of the algorithm.

In ATNoSFERES-II, a drastic strategy is used in such a case: when no edge is eligible, the evaluation of the agent stops and *FAIL* is returned. As a consequence, the remaining number of time steps for the current test is decreased to zero, and the fitness for this test is null (the overall fitness is the sum over several tests, see section 4). In the following, this choice of abruptly ending the test when no edge is eligible will be referred to as *Finish*, while the random choice of ATNoSFERES will be called *Random*.

We must emphasize that the third source of non-determinism (performing a random action when crossing an edge with no action label) seems experimentally to have much less impact on non-deterministic performances. We observed that the few such edges still present in the population after some generations were never eligible.

3.3 Contradiction filtering

In the original version of ATNoSFERES, an edge of the control graph could eventually be labeled with contradictory conditions, resulting in its permanent ineligibility. Such a situation cannot happen in LCSs, since the condition part of a classifier cannot contain contradictions. In the present

work, the occurrence of contradictory conditions in the label of edges is prevented by forbidding more than one condition specifying the value of the same attribute in the condition: once an attribute has been used in a given label, all subsequent tokens involving that attribute are ignored for that label. The use of this mechanism will be referred to as *No-Contradiction*, in contrast with the *Contradiction* original ATNoSFERES algorithm.

3.4 Behavior of swap/roll/unroll tokens

In the original version of ATNoSFERES, the swap, roll and unroll stack tokens could operate without discrimination on any data in the stack, whereas some other stack tokens could only operate either on the nodes or on the labels: e.g. *dup node* only operated on nodes, and *dup label* only on labels. This "typing" of tokens is extended in ATNoSFERES-II to all available stack tokens, in order to facilitate structural manipulations of the automata being built. The previous set of the untyped swap/roll/unroll tokens will be referred to as *all*, while the typed ones will be referred to as *node/label*.

4. EXPERIMENTAL SETTINGS

4.1 Representation

Our experimental set-up is the same as in [4], described in section 2.1, except for the new features of ATNoSFERES-II. We used a 1% bias for mutation probabilities. In particular, since one of the goals of the present experiments is to compare the uniform and the bit-flip mutations on a fair basis, the distribution of the set of tokens is also the same (see table 1), except that, of course, integer- and not bit-strings are used in the *Uniform* approach of ATNoSFERES-II (section 3.1). The other difference, as pointed out in section 3.4, lies in the use of "typed" *swap*, *roll* and *unroll* tokens (see alternatives in table 1), when it comes to compare *Node/Label* against *All*.

Capitalizing on previous published experiments, we do not start with as many genome lengths as before [5, 4]. Some preliminary tests with lengths of 250, 300 and 350 tokens on a subset of parameter combinations demonstrated that, in this range, the genome lengths had no statistically significant impact on the final performance. From thereon, and with an abusive generalization to all environments, we shall consider that the average results obtained with this limited range of lengths can be extended to other lengths for all environments.

Also, in the next section, the graphical representation of the graphs is different from that of [5, 4], in order to make the figures more readable (e.g. see figure 2). First, the condition part is presented above the action part. Then, each condition is here prefixed by a letter representing the perception of the agent: **f**ood, **t**ree, or **e**mpty. The *Start* node is labeled *0*, and the *End* node is the node labeled with the largest number. If no condition is present on an edge, any perception will match. Nodes and edges are represented as usual by circles and directed arrows.

4.2 The environments and the fitness

Three different environments, respectively *Maze10* [6], *E1* and *E2* [9], have been used to validate and test the modifications of ATNoSFERES proposed in section 3. For all these environments, the "optimal policy" is the best policy that can be found with the standard limited perception used in the so called "woods" experiments [13] without any limit on the number of memory bits. The number of steps to find the food from each cell given by this optimal policy has been presented in [4].

For each given environment, each automaton is started once in each cell, and its fitness is measured as the average number of steps it takes to find the food, averaged over all cells (to be minimized).

5. RESULTS

Because we did not want to make any assumption about the usefulness of each on the modifications to the original ATNoSFERES proposed in section 3, systematic experiments were conducted in each of the three environments in order to evaluate the 16 possible combinations of each of the four modifications, with 50 independent runs per setting.

5.1 Best solutions found

This section will present the best overall results obtained in each environment. In each case, both the automaton and a picture of its performance with respect to the optimal policy is given. All those automata have been obtained using the *Uniform* and *No contradiction* modifications – the other two will be detailed in each subsection.

5.1.1 Maze10

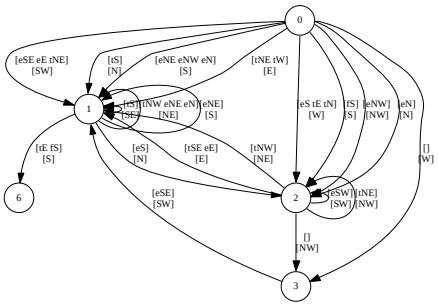

Figure 2: Best automaton found for the Maze10 environment (5.11 steps to food; an optimal policy requires 5.05). 19 edges (3 nodes) that are never elected (reached) are not represented, for the sake of readability.

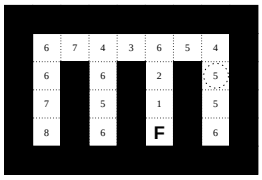

Figure 3: Performance of the best policy found for the Maze10 environment (by the automaton of figure 2). It is only one step worse, and on a single cell, than the optimum (dotted circle). The rest is optimal.

In Maze10, the best result of ATNoSFERES-II (figure 2) has a policy that is is only one step longer than the optimal policy (see figure 3), when the *Start* cell is just below the

NE corner of the maze. It was found using 300 codons, and the *Node/Label* and *Finish* parameters. As far as we can tell, no LCS has ever performed so well on this problem. The average number of steps to food is 5.11, vs. 5.61 for ATNoSFERES in [4].

5.1.2 E1

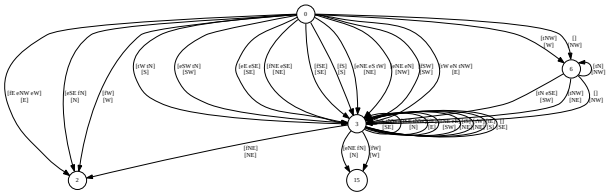

Figure 4: Best automaton found for the E1 environment (2.90 steps to food; an optimal policy requires 2.81). 30 edges (11 nodes) that were never elected (reached) are not represented, for the sake of readability.

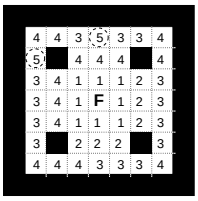

Figure 5: Best policy found for the E1 environment (by the automaton of figure 4). Only two cells are only two time steps away from the optimum (dashed circles). The rest is optimal.

In environment E1, the best policy found is only 4 steps longer than the optimal one (figure 5). It was found with a chromosome length of 250, and using the *All* and *Finish* settings. Again, as far as we are aware, no LCS has ever performed so well on this problem: The average number of steps to food is 2.90, vs. 3.3 for ACS in [9].

5.1.3 E2

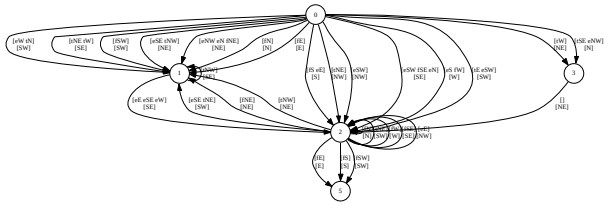

Figure 6: Best automaton found for the E2 environment (3.29 steps to food; an optimal policy requires 2.98). 13 edges (1 node) that were never elected (reached) are not represented, for the sake of readability.

In environment E2, the best policy found is 15 steps away from the optimum (figure 7). Its genotype length is 350, and again the *All* and *Finish* strategies were used. As for the two

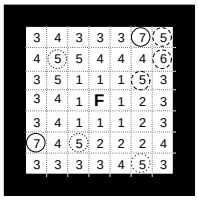

Figure 7: Best policy found for the E2 environment (by the automaton of figure 6). Three cells are one time step away from the optimum (dotted circles), three cells are two time steps away from the optimum (dashed circles), and two cells are three time steps away from the optimum (solid circles). The rest is optimal.

other environments, as far as we can tell, no LCS has ever performed so well on this problem: The average number of steps to food is 3.29, vs. 3.58 for ATNoSFERES in [4].

5.2 Validation of the different modifications

The noticeable improvements of the off-line best results reported in the previous section when using ATNoSFERES-II have been obtained using (some of) the modifications presented in section 3. The present section investigates which of the corresponding modifications has a significant impact on the overall performance of the algorithm in average.

	Maze10	E1	E2
Uniform	8.2	3.7	4.3
BitFlip	12	5	6.4
p-value	$\approx$ **0**	$\approx$ **0**	$\approx$ **0**
All	10	4.4	5.4
Node/Label	9.7	4.4	5.3
p-value	**0.02**	1.0	0.9
Contradiction	10	4.4	5.4
No contradiction	9.8	4.3	5.3
p-value	0.2	**0.05**	0.4
Random	9.7	4.3	5.3
Finish	10	4.4	5.4
p-value	**0.04**	0.6	0.2

Table 2: Average number of steps to food for each parameter value and environment, and T-test p-value for each couple of alternative parameter values. A p-value lower than 0.05 means a statistically significant difference (in bold).

Table 2 analyzes the results according to each couple of parameter values, that is, for each value of a given parameter (e.g. *Uniform* or *BitFlip*), the results are averaged over the 8 possible values of the other 3 parameters. It clearly appears that *Uniform* mutation produces significantly better results than *BitFlip* in all three environments. The statistical significance is not so clear for the other strategies, except for *Node/Label* and *Random* for Maze10, and *No contradiction* for E1.

Hence in the remaining of this section, *Uniform* is assumed, and the analysis is carried over on the 8 sets of experiments for which *Uniform* was used. Table 3 shows the results of this analysis, and it appears that *Random* gives significantly better results for Maze10 and E2, but not for E1. Other parameters do not produce significantly different results, with the exception of *Node/Label* for Maze10.

Uniform	Maze10	E1	E2
All	8.6	3.7	4.3
Node/Label	7.7	3.7	4.4
p-value	$\approx$ **0**	0.7	0.6
Contradiction	8.1	3.8	4.4
No contradiction	8.3	3.7	4.3
p-value	0.4	0.06	0.2
Random	7.9	3.7	4.3
Finish	8.4	3.7	4.4
p-value	**0.01**	0.9	**0.03**

Table 3: Average mean for each parameter value and environment, and T-test p-value for each couple of alternative parameter values, assuming Uniform.

Uniform & Random	Maze10	E1	E2
All	8.1	3.7	4.2
Node/Label	7.7	3.8	4.3
p-value	0.1	0.5	0.4
Contradiction	8	3.8	4.3
No contradiction	7.8	3.6	4.2
p-value	0.5	**0.03**	0.4

Table 4: Average mean for each parameter value and environment, and T-test p-value for each couple of alternative parameter values, assuming Uniform and Random.

Assuming now *Uniform* and *Random*, *No contradiction* still gives significantly better results for the E1 environment (see table 4). So even if the *Contradiction/No contradiction* parameters do not produce significantly different results for all environments, *No contradiction* nevertheless gives better average mean results in all experiments, whether considered alone, or in conjunction with either *Uniform* alone or *Uniform* and *Random*. Furthermore, all the best known solutions found for the tested environments use the *No contradiction* value.

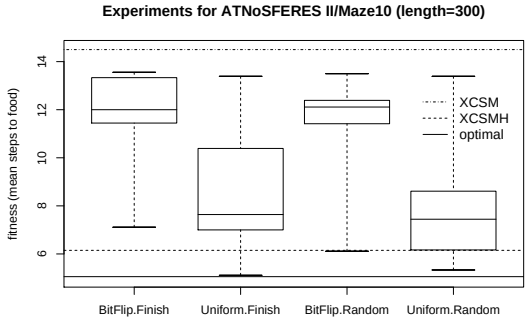

Figure 8: Results for the Maze10 environment, according to the mutation and default node action variables. See section 5.2 for details and analysis.

Box plots (see figures 8, 9 and 10) are provided to give an informal idea of the distributions of the fitnesses for each environment, according to the two meaningful variables (mutation and default node action). These graphics represent meaningful statistics rather than all the data: the median as an horizontal line in the boxes, rather than the sample mean, as the measure of central tendency due to the skew

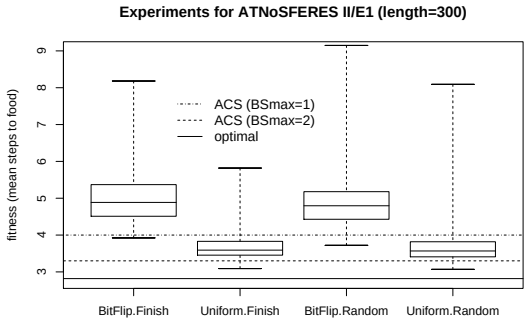

Figure 9: Results for the E1 environment, according to the mutation and default node action variables. See section 5.2 for details and analysis.

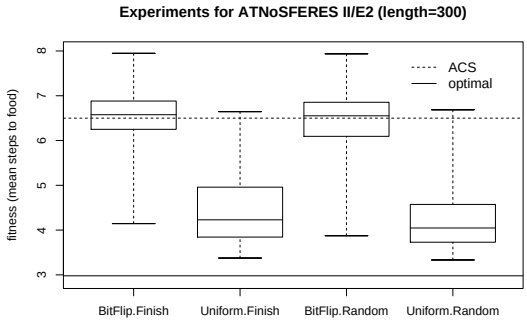

Figure 10: Distribution of the results for the E2 environment, according to the mutation and default node action variables. See section 5.2 for details and analysis.

in the distribution caused by the fixed lower limit of fitness values; the second and third quartiles, taken as the upper and lower edges of the boxes; finally, the "whiskers" are plot from both quartiles to the extremal values.

The distribution for ATNoSFERES corresponds more or less to the BitFlip/Random case. From these plots, one can see that, on the one hand, Uniform mutation clearly outperforms BitFlip in all cases, and on the other hand, for example for Maze10 (see figure 8), that approximately 25% of the Uniform/Random solutions outperform XCSMH (1st quartile).

5.3 Comparison of convergence time with ACS and XCSMH

Table 5, borrowed from [4], gives a comparison between the performance of the original ATNoSFERES and that of some well-known LCSs from the literature in terms of convergence time.

From this table, it was clear that, though being more general and obtaining sometimes better solutions, ATNoSFERES was several orders of magnitude slower than all other tested LCSs. However, since the modifications that lead to ATNoSFERES-II significantly improved those performances, it remains to check again whether this improvement has a significant impact on the running time of the algorithm, and to compare this running time to that of the

envir.	LCS type	perf.	LCS	$PR(\%)$	NG	NT	NT/LCS
Maze10	XCSM	15.1	7,000	100	8.42	45,000	6.42
Maze10	XCSMH	6.1	6,500	7.30	4909	360.10^6	55,000
E1	ACS ($BS_{max} = 1$)	4	4,400	30.92	5115	218.10^6	50,000
E2	ACS ($BS_{max} = 2$ or 3)	6.5	2,000	83.84	835	14.10^6	7,000

Table 5: Comparison between the original ATNoSFERES and LCSs. The measure is the average number of trials needed to reach the performance given in column "perf.". PR is the percentage of runs where ATNoSFERES outperforms the corresponding LCS, and NG is the average number of generations before this takes place. Thus, with 300 individuals per generation, the average number of evaluation runs necessary to outperform the corresponding LCS is $NE = 300 * NG * 100/PR$, and the average number of elementary runs NT for an environment with NS start cells ($NS = 18$ for Maze10, 44 for E1 and 48 for E2) is: $NT = NS * NE$. Thus NT gives the average number of elementary runs needed by ATNoSFERES to outperform the corresponding LCS. Finally, NT/LCS gives a good approximation of the factor by which the corresponding LCS is faster than ATNoSFERES to reach its best performance. Note that no performance comparison is given on E1 against ACS with $BS_{max} = 2$ since ATNoSFERES never outperforms it.

same LCSs. The results are shown in table 6, and show that ATNoSFERES-II is indeed much faster than ATNoSFERES, though still slower than the other LCSs.

6. DISCUSSION

The first remark we want to make is that, despite the improvement in performance brought by our modifications, we still did not succeed in reaching the optimal policy in any of the three environments tested here. This illustrates how difficult these simple non Markov environments are for a genetic-based machine learning system.

From section 5.2, we can see that the most important and significant improvement is the change of representation, through the induced change of mutation. The impressive efficiency of this modification is probably mainly due to the codons-to-tokens mapping that was used in the original version of ATNoSFERES (table 1). As a matter of fact, one can note that the leftmost bit is an indicator for label tokens: tokens enumerated from 100000 to 111111 are exclusively action or condition tokens, to be pushed on the stack. Therefore, when using a bit-flip mutation over the binary encoding of the codons, with a 1% probability, once a codon has its leftmost bit set, it has 99% probabilities to remain an action or a condition: the combination of the mapping and the bit-flip mutation resulted in a very strong bias toward labels exploration, with very little structure exploration. On the other hand, when using the uniform mutation over string of integers, structural changes occur with probability 0.5 during mutation – and the results show that this additional structural exploration significantly improves the performance of the algorithm. Further work will look in detail at the statistics of every token used by the best solutions, and should help to find a more efficient distribution bias over the different tokens, that will be easy to implement in the integer representation.

The introduction of *Finish* default action node did not prove very useful: either *Random* performs significantly better, or the difference is not statistically significant. But this conclusion is consistent with the general consensus in the Reinforcement Learning community, according to which some non-determinism is always helpful when tackling Non Markov problems, as purely deterministic controllers might easily get stuck in local minima. However, in the context of evolutionary techniques, non-determinism makes the evaluation more difficult and may be detrimental to the convergence of the GA. Some optimal trade-off probably remains to be found here.

The *Node/Label* set of stack tokens, introduced in order to facilitate structural modifications of the automaton being built during the interpretation process, only improves results in the Maze10 environment. Restricting to the Uniform/Random combination, this advantage vanishes (see table 3). More precisely, on Maze10, only restricting to *Uniform* results in *Random* still significantly outperforming *Finish* on average (see table 3), and such is the case too for *Node/Label* with respect to *All*, with even more significance. So, the incompatibility on Maze10 lies between *Node/Label* and *Random*.

Nevertheless, this phenomenon is difficult to explain. In [4], it was shown that the kind of structures necessary to perform optimally in Maze10, E1 and E2 are very different from one another. It is likely that a difference in the token language would result in a different probability of obtaining this or that structure, but the corresponding analysis is at the moment beyond our reach. If this assumption appears to be true, this means that the token language itself has to be specifically tuned for each particular problem.

Quite surprisingly, the *Contradiction/Non-Contradiction* alternative does not systematically make significant difference either. As a matter of fact, the difference is significant in average fitnesses for both alternatives only on E1. It could have been expected that biasing the population towards meaningful condition parts would systematically reduce the search space towards efficient automata, but this does not seem to be the case, even if the Non-Contradictory language seems to perform slightly better in all experiments. Here, we must take into account the fact that the Non-Contradictory language induces more introns in the genotype, as many conditions are simply ignored once put upon an edge. So again, there is probably a trade-off between a more efficient search, thanks to more meaningful edges, and degradation of performances because of the genotype bloat. Future work will investigate this assumption by trying much shorter genotypes, leaving less room to introns, and/or adding some parsimony pressure to the fitness. If our assumption is true, we should observe even better results on average when exploring a space biased towards shorter genotypes.

Finally, from the results of section 5.3, it is clear that, even if the improvement in performance has a serious impact

envir.	LCS type	perf.	LCS	$PR(\%)$	NG	NT	NT/LCS
Maze10	XCSM	15.1	7,000	100	10	54,000	7.7
Maze10	XCSMH	6.1	6,500	27.5	5,300	104.10^6	16,000
E1	ACS ($BS_{max} = 1$)	4	4,400	92	2,500	36.10^6	8,100
E1	ACS ($BS_{max} = 2$)	3.3	1,200	22.5	6,900	40.10^7	337,000
E2	ACS (BS_{max} 2 or 3)	6.5	2,000	99	460	67.10^5	3,300

Table 6: Same comparison as in table 5 between the new version of ATNoSFERES (assuming only Uniform mutation and Random default node action), and LCSs. This time, performance comparisons can be given on E1 against ACS with $BS_{max} = 2$.

on convergence time, this impact is still far form sufficient to make ATNoSFERES-II competitive in speed with ACS, [11, 9] and XCSMH [8], the most efficient Michigan-style LCSs (the case of XCSM can be ignored due to the much higher efficiency of XCSMH). We feel that this conclusion advocates once again for the more general claim that:

- the Pittsburgh approaches are often more robust than Michigan ones and, given enough time, can often obtain better performances, but
- they are generally much slower, which results in poorer performance under strong CPU time constraints.

7. CONCLUSION

In this paper, we have studied the effect on the performance of several modifications of the Pittsburgh style system ATNoSFERES. Some of these modifications, such as a different encoding of the token language, have been demonstrated to have a statistically significant impact on the performance. Other modifications, such as the deterministic versus stochastic behavior of the automaton, or the presence versus absence of contradictions on the label of edges, have given less conclusive results.

In some cases, the effect of these variations have not been clearly explained so far: further investigations are necessary to change the assumptions we made about these phenomena into unquestionable explanations.

Anyway, one clear result of this paper is that, thanks to some of these modifications, ATNoSFERES-II was able to find the best CS-based controllers known so far in the Maze10, E1 and E2 environments. However, it should be noticed that this improvement in performance was not accompanied by a clear improvement in convergence speed. This last assertion, as well as general considerations known as the Michigan vs Pittsburgh debate, are a strong incentive to try to design a Michigan-style system based on the same representation as ATNoSFERES. We hope that the studies conducted here will help in making this future system more efficient.

8. REFERENCES

[1] S. Landau and S. Picault. Stack-Based Gene Expression. Technical report LIP6 2002/011, LIP6, Paris, 2002.

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

[3] S. Landau, S. Picault, O. Sigaud, and P. Gérard. A comparison between ATNoSFERES and XCSM. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 926–933, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[4] S. Landau and O. Sigaud. A Comparison between ATNoSFERES and LCSs on non-Markov problems *(to appear). Information Sciences*, 2004.

[5] S. Landau, O. Sigaud, S. Picault, and P. Gérard. An Experimental Comparison between ATNoSFERES and ACS. In W. Stolzmann et al., editors, *IWLCS-03. Proceedings of the Sixth International Workshop on Learning Classifier Systems*, LNAI, Chicago, july 2003. Springer.

[6] P.-L. Lanzi. An analysis of the memory mechanism of XCSM. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 643–651, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Morgan Kaufmann.

[7] P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, editors. *Special Issue of* Evolutionary Computation, volume 11:3. MIT Press, 2003.

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

[9] M. Métivier and C. Lattaud. Anticipatory Classifier System using Behavioral Sequences in Non-Markov Environments. In Stolzmann et al. [12], pages 143–163.

[10] M. Nicolau, A. Auger, and C. Ryan. Functional dependency and degeneracy: detailed analysis of the GAuGE system. In *Evolution Artificielle 03*, pages 15–26. LNCS 2936, Springer Verlag, 2003.

[11] W. Stolzmann. Latent Learning in Khepera Robots with Anticipatory Classifier Systems. In Wu [16], pages 290–297.

[12] W. Stolzmann, P.-L. Lanzi, and S. W. Wilson, editors. *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS'02)*, LNAI, Granada, september 2002. Springer-Verlag.

[13] S. W. Wilson. Classifier Systems and the Animat Problem. *Machine Learning*, 2(3):199–228, 1987.

[14] S. W. Wilson and D. E. Goldberg. A critical review of Classifier Systems. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 244–255, Los Altos, California, 1989. Morgan Kaufmann.

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

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