ATNoSFERES : a Model for Evolutive Agent Behaviors

Samuel Landau; Sébastien Picault; Alexis Drogoul
LIP6; case 169; 4 place Jussieu, 75252 Paris Cedex 05, FRANCE
{Samuel.Landau,Sebastien.Picault,Alexis.Drogoul}@lip6.fr
http://www-poleia.lip6.fr/~{landau,picault,drogoul}

Abstract

This paper introduces ATNoSFERES, a model aimed at designing evolutive and adaptive behaviors for agents or multi-agent systems. We first discuss briefly the main problems raised by classical evolutionary models, which are not intended to produce agents or behaviors but to solve problems. Then we provide detailed explanations about the model we propose and its components. We also show through a simple example how the system works, and give some experimental results. Finally, we discuss the features of our model and propose extensions.

Keywords

Genetic encoding, evolutionary computing, adaptive behaviors, agents, artificial life, ATN[1]

1 Introduction: limitations of existing models

ATNoSFERES (Picault and Landau, 2001) is a model for designing evolvable agent behaviors, possibly with agents operating at different organization levels. The behavior of the agents of each level may either be given, or be *built* from an hereditary substrate (a bit string called "chromosome"). This model is an attempt to overcome problems raised by classical evolutionary paradigms in the design of *behaviors*.

Genetic algorithms (Holland, 1975), (Goldberg, 1989), (Bäck and Schwefel, 1993) allow non-semantic manipulations of the genotypes (strings of bits), inducing in most cases gradual modifications in the phenotypes. Thus they ensure that individuals produce children with quite the same level of adaptation to the environment than their own. However, genetic algorithms have a very poor expressive power, since their purpose is the optimization of a set of parameters in behaviors which *have to be given a priori*.

On the contrary, the genetic programming paradigm (Koza, 1992), (Gruau, 1994) is based on the evolution of *programs* (i.e. instructions trees), thus their expressive power is much higher. But the genetic operators associated with trees operate in a syntactic way on semantic structures, so they induce strong variations in the effects of the resulting program ; in addition, the impact of the genetic operators tightly depends on the level at which they operate (a modification near the root is likely to have a deeper influence than one on a leaf). Moreover, the behavior of an agent cannot be reduced to a program.

The *Typogenetics* (Hofstadter, 1979) is an interesting attempt to build a "life-like" formal system: strings are translated into "typoenzymes" that operate on the existing strings to produce new ones. There are no genetic operators in such a system – it works and evolves *autonomously*. However, on the one hand the translation of the strings and the production of new strings are pure deterministic processes (this can obviously become harmful when trapped into a loop); on the other hand, the system is not intended to be "used" for producing behaviors: its purpose is only the production of strings.

Artifical life models such as *Tierra* (Ray, 1990) try to investigate the possibilities given by the self-replication of programs in a virtual world. In Tierra this world is a virtual machine producing random errors ; the programs replicate by treating their own instructions as data. Life-like phenomena (parasitism for instance) have been observed; however, there is no control at all over the evolution of the system, and the behaviors of these virtual organisms are only directed by their survival in the "memory" of the machine, i.e. in a very specific environment.

A first attempt to escape the behavioral limitations of genetic algorithms, and to reinforce the autonomy of the expression of the genotype without being caught into a useless virtual world, has been implemented in the EDEN model (Picault et al., 1997). In that model, a string is translated into a list of instructions that are used as a program, called an "enzyme". These enzymes might have very diverse functions (depending on the specification of their "programming language"), among them, on the one hand the translation of the string into new enzymes, and on the other hand genetic operations on the string (random mutations, deletions, insertion of substrings, replications, etc.). The main obstacle in such a system is the high sen-

[1] ATN stands for Augmented Transition Network

sibility of the enzymes to mutations, like in genetic programming. This issue is crucial since the translation of the string is performed by an enzyme (built from a gene in the string, which is subject to mutations).

2 Description of our model

2.1 General principles

ATNoSFERES uses the SFERES framework (Landau et al., 2001) as a tool for modelling the agents classes, integrating those classes to the system, designing an environmental simulator and providing classical evolutionary techniques.

In particular, it provides a general class, the `ATNAgent`, the behavior of which is produced through the following steps:

1. a translator produces tokens from the string,
2. an interpreter uses these tokens as instructions to build a graph (an ATN[1] (Woods, 1970)),
3. finally, the graph determines the behavior of the agent.

The translator and the interpreter themselves are agents; in the following lines, we will consider that their behavior is given, but it could evolve as well (like in EDEN) to provide the system with higher autonomy.

2.2 The ATN Graph and the `ATNAgent`

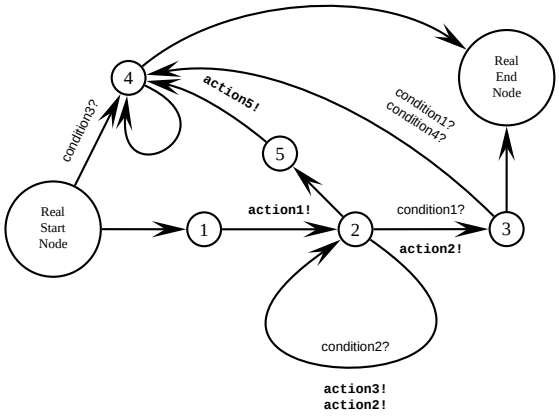
The `ATNAgent` class is intended to behave according an ATN graph. ATN have previously been used by Guessoum (1996) for designing agent behavior. Each subclass of `ATNAgent` is associated with two collections of tokens: condition ones and action ones. The actions are behavioral "primitives" that can be performed by the agent, the conditions are perceptions or stimuli that induce action selection. The edges of the graph are labeled with a set of conditions and a sequence of actions (see figure 1).

The ATN built by adding nodes and edges to a basic structure containing two nodes: a "Real Start Node" and a "Real End Node". At each time step, the agent (initially in the "Real Start Node" state) randomly chooses an edge among those having either *no condition* in their label, or *all conditions simultaneously true*. It performs the actions associated with this edge and jumps to the destination node. It stops working when its state is the "Real End Node".

2.3 The Interpreter

The purpose of the interpreter is to build an ATN from tokens. Some of these tokens will be action or condition ones that are used to label edges between nodes in the

Figure 1: An example of ATN.



ATN. The other ones are interpreted as instructions, either to create nodes or connect them, or to manipulate the structure under construction.

As we mentioned in section 1, the structure built by the tokens sequence has to be robust towards mutations. For instance, the replacement of one token by another, or its deletion, should have only a *local impact*, rather than transforming the whole graph. Therefore, we use a "top-level" programming language operating on a list (see table 1).

Table 1: The ATN-building language.

token (initial list state)		(resulting list)
condition? $(x \ldots)$	$\longrightarrow$	(condition? x ...)
`action!` $(x \ldots)$	$\longrightarrow$	(**`action!`** x ...)
node $(x \ldots)$	$\longrightarrow$	$(N_i\ x \ldots)^a$
startNode $(x \ldots)$	$\longrightarrow$	$(N_i\ x \ldots)^b$
endNode $(x \ldots)$	$\longrightarrow$	$(N_i\ x \ldots)^c$
connect (c2? $x\ N_i\ y$ c1? z **`a2! a1!`** $t\ N_j\ u \ldots$)	$\longrightarrow$	$(x\ N_i\ y\ z\ t\ N_j\ u \ldots)^d$
dup $(x\ y \ldots)$	$\longrightarrow$	$(x\ x\ y \ldots)$
dupObject $(x\ y\ N_i\ z \ldots)$	$\longrightarrow$	$(N_i\ x\ y\ N_i\ z \ldots)$
popRoll $(x\ y \ldots z)$	$\longrightarrow$	$(y \ldots z\ x)$
pushRoll $(x \ldots y\ z)$	$\longrightarrow$	$(z\ x \ldots y)$
swap $(x\ y \ldots)$	$\longrightarrow$	$(y\ x \ldots)$
forget $(x\ y \ldots) + [z \ldots]$	$\longrightarrow$	$(y \ldots) + [x\ z \ldots]^e$
recall $(x \ldots) + [y\ z \ldots]$	$\longrightarrow$	$(y\ x \ldots) + [z \ldots]^e$

[a] creates a node N_i
[b] creates a node N_i and connects "RealStartNode" to it
[c] creates a node N_i and connects it to "RealEndNode"
[d] creates an edge between N_j and N_i, with (c1?& c2?) as condition label and the list {**`a1!`**,**`a2!`**} as action label
[e] with an auxiliary stack

If an instruction cannot execute successfully, it is simply ignored, except instructions operating on nodes (i.e. *connect* and *dupObject*) which are "pushed" in the list until new nodes are produced; then they try to execute

again with the new data. Finally, when the interpreter does not receive tokens any more, it terminates the ATN: actions and conditions tokens still present between nodes are treated as *implicit connections* (so that new edges are created) and the consistency of the ATN is checked ("Real Start Node" is linked to nodes having no incoming edges, except from themselves; in the same way, nodes having no outgoing edges are linked to "Real End Node").

2.4 The Translator

The translator has a very simple behavior. It reads the genotype (a string of bits) and decodes it into a sequence of tokens. It uses a *genetic code*, i.e. a function

$$\mathcal{G} : \{0,1\}^n \longrightarrow \mathcal{T} \qquad (|\mathcal{T}| \leq 2^n)$$

where $\mathcal{T}$ is a set of tokens, which includes both action and condition ones (specific to the agent to build) and those understood by the interpreter (see table 1).

Depending on the number of tokens available, the genetic code might be more or less redundant. If necessary, it can be designed in order to resist mutations, but we will not discuss this issue in this paper.

3 Experiments

The purpose of this section is both to demonstrate how the system works by using it on a simple example (with a single class of agents operating at the same level), and to provide some results regarding the behaviors that evolved in the following experiments.

3.1 Experimental setup

To illustrate the evolution of simple behaviors, let us consider an experiment with a discrete environment containing a color light bulb and a single agent (instance of a subclass of the general `ATNAgent`) with the action and perception abilities described in table 2. We want the agent to go to the right when the light is green and to the left when it is red. To make the agent behavior evolve, we apply the rules of darwinian selection over a population of 100 homogeneous agents.

Table 2: Action and condition tokens of the agent.

	Actions		**Conditions**
N!	no action		
R!	move to the right	g?	true when the light is green
L!	move to the left	r?	true when the light is red
U!	move up	rand?	true with probability $p = 0.5$
D!	move down		

The genetic code for these agents contains the 11 interpreter tokens and the 8 action/condition tokens; thus it needs at least 32=2^5 codons. In the following experiments, a 32-codon genetic code has been automatically built from a circular list containing the interpreter and action/condition tokens. The chromosome of the agents is initially a *random* bit string with a *random length* (from 200 to 300 bits).

We evaluate the fitness of each agent by making it run during 100 time steps in its environment. The color of the light bulb randomly flips (with probability 0.05 at each time step) from green to red and vice-versa. The rewarding rules in the fitness function are: +1 point if the move is correct, -1 point if it is erroneous (e.g. left when green), 0 in the other cases (e.g. move up). Only the first move performed during the current time step is rewarded ("do nothing" is not considered as a move).

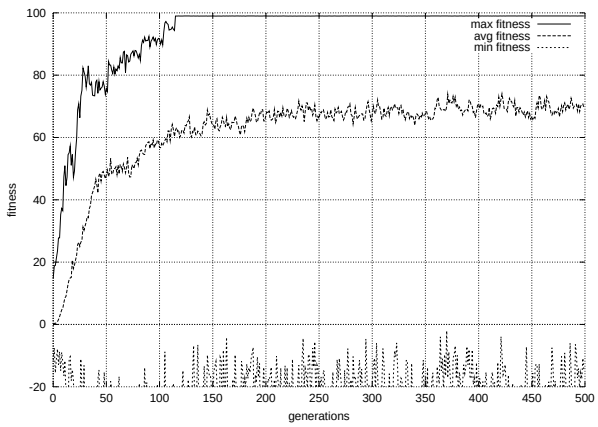
At each generation, the agents are evaluated through their average fitness (calculated over 10 runs in the above conditions) and selected to produce 30 new agents (by crossing over chromosomes), thus replacing 30 agents removed from the old population (depending on their fitness, too).

We have experimented with various mutation strategies, among them:

1. before their evaluation, *all agents* are subject to punctual random mutations of their chromosome with rate r (r % of the bits are randomly flipped);
2. same situation, but p % of the mutations are random insertions or deletions of codons in the chromosome, instead of punctual mutations.

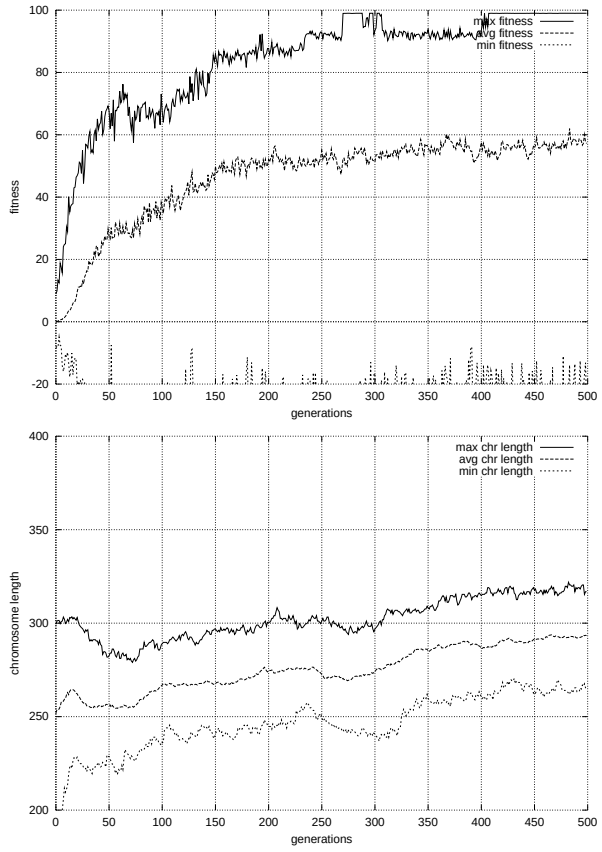
3.2 Results

Figure 2: Evolution of the fitness (punctual mutations only, r = 1 %).



As agents are initialized in their "Real Start Node" state, the first time step is used to jump to one of the available nodes. Then, during the 99 other time steps, the be-

Figure 3: Evolution of the fitness and chromosome length (random insertions and deletions, r = 1 %, p = 20 %)



havior of the agents only depends from their ATN structure and its ability to respond to environmental changes. Thus, the maximum fitness in these experiments is 99 (correct answers at each time step after the first one).

Figure 2 shows the average evolution of the fitness with the first mutation strategy. It has been calculated from 10 experiments. Figure 3 shows the average evolution of the fitness with the second mutation strategy, and the evolution of the chromosome length (10 experiments too). In a few cases (almost in the first strategy), no good solution is found before 500 generations.

4 Discussions

Though the problem to solve is very simple, these results validate the ATNoSFERES model for evolving agent behaviors from a fine-grain substrate. But we would like to focus on the specificity of this model and give a qualitative analysis of the behaviors produced by natural selection.

The ATN described on figure 4 is the "optimal" solution to this problem (99 points with the simplest ATN structure): when the light is red, go to the left; when it is green, go to the right. To produce this ATN, only 35 bits are theoretically required, for example to encode the following tokens:

node, g?, **R!**, *dupObject*, **L!**, r?, *dupObject*

(with a maximal use of implicit connections). But in this solution the order and nature of tokens is crucial, thus it is highly vulnerable to mutations. In addition to this, the agents have much more bits in their chromosome than necessary – this can be a source of inadequate behavior.

Figure 4: The "optimal" ATN (providing the highest fitness with the simplest structure).

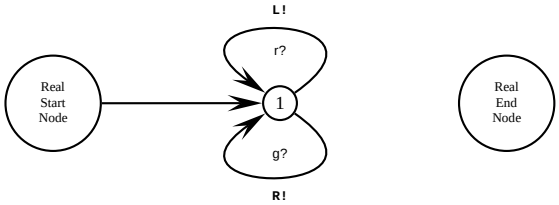
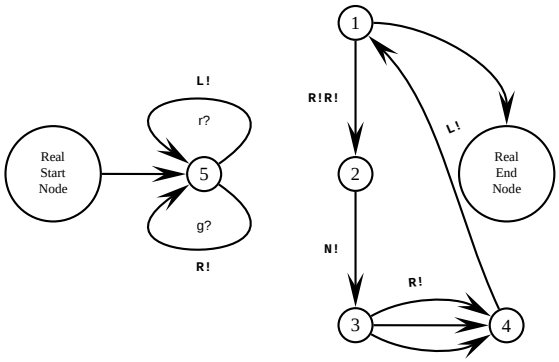


Figure 5: An ATN built by natural selection, implementing the best behavioral strategy.



The experimental results show that two strategies are used to produce an adequate behavior (99 points). The first one consists in building a simple ATN (very close or same than the "optimal" ATN on figure 4), by delaying the node creation and thus using tokens that have no effect. For instance, a 207-bit chromosome encoding the following tokens:

L!, *dup*, *swap*, *popRoll*, *popRoll*, *forget*, *popRoll*, *swap*, *forget*, *recall*, *recall*, **R!**, **L!**, *recall*, rand?, **L!**, *forget*, *startNode*, **R!**, *dup*, g?, *dupObject*, *popRoll*, *pushRoll*, **L!**, *forget*, *pushRoll*, **L!**, r?, *startNode*, *forget*, *dup*, *dupObject*, r?, **R!**, *forget*, *dup*, **R!**, *dup*, g?, *popRoll*

produces an ATN very close to figure 4, with labels (r?, **L!L!R!R!**) on one edge and (g?, **R!R!**) on the other. This is a good example of using the properties of the ATN-building language.

The agents using the second strategy build a complex ATN in which a small subset only is used, for instance like

the ATN on figure 5. It leads to exactly the same behavior than the "optimal" solution, since a large part of it cannot be reached from the other nodes.

5 Conclusions and perspectives

We have presented ATNoSFERES, a model for evolutive agents behaviors.

As an evolutive approach, ATNoSFERES has two main features. First, it separates the genetic information structure (plain bit string, the lexical structure) from its interpretation (ATN, the semantic structure). Thus, the semantic structure built is *always correct*. The behavior described by the ATN always has a meaning – even if it is not adequate to the environment. Not having to worry about the syntactic correctness of the automatically designed semantic structure is a good point over many other evolutive approaches.

The second main evolutive features are related to the genetic operators. The level of influence of the classical genetic operators – mutation and crossover – does not depend on the parts of the bit string they involve (neither on their location in the bit string nor on their size). This is also a main advantage over many evolutive approaches. As a matter of fact, mutations only have a local impact in the expression of the genetic information, and crossovers involve bit substrings which carry locally functional genetic code. We might also consider more exotic genetic operators, such as deletions/insertions in the bit string. These operators in particular permit to smoothly manage string resizing, since they only have a local impact in the ATNoSFERES model.

As a model for designing agents, ATNoSFERES does not set any restriction on the agent level specification. Furthermore, agents can be introduced later on at a lower organization level (for instance inside an agent), keeping the latter structure, if a finer-grain agent specification is needed.

Finally, as a model for automatic behavior design, ATNoSFERES provides a simplified framework, where only the conditions and actions of the agents have to be specified. Then, with an ATN as the structure for behavior description, it is possible to directly describe and explain the behavior of any agent.

In further works, we plan to introduce a metabolic regulatory mechanism, associated with the actions within the ATN. It will act as an environmental constraint, allowing or disabling some edge transitions at a given time.

References

Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

David Edward Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Pub. Co., 1989.

Frédéric Gruau. *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*. PhD thesis, ENS Lyon – Université Lyon I, january 1994.

Zahia Guessoum. *Un environnement opérationnel de conception et de réalisation de systèmes multi-agents*. PhD thesis, LIP6 – Université Paris VI, may 1996.

Douglas Hofstadter. *Gödel, Escher, Bach: an Eternal Golden Braid*. Basic Books, Inc., New York, 1979.

John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor: University of Michigan Press, 1975.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts, 1992.

Samuel Landau, Stéphane Doncieux, Alexis Drogoul, and Jean-Arcady Meyer. SFERES, a Framework for Designing Adaptive Multi-Agent Systems. Technical report, LIP6, Paris, 2001.

C. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors. *Artificial Life II*, London, 1990. Addison-Wesley.

Sébastien Picault and Samuel Landau. ATNoSFERES : a Darwinian Evolutionary Model for Individual or Collective Agent Behavior. Technical report (in press), LIP6, Paris, 2001.

Sébastien Picault, David Servat, and Frédéric Kaplan. EDEN : un système évolutif endosémantique. Technical report, École Nationale Supérieure des Télécommunications, Paris, septembre 1997.

Thomas S. Ray. An evolutionary approach to synthetic biology, zen and the art of creating life. In Langton et al. (1990).

W. A. Woods. Transition network grammars for natural language analysis. In Grosz, Jones, and Webber, editors, *Readings in Natural Language Processing*. 1970.