SOS++: Finding Smart Behaviors Using Learning and Evolution

Bertrand Mesot[1], Eduardo Sanchez[1], Carlos-Andres Peña[1] and Andres Perez-Uribe[2]

[1]Swiss Federal Institute of Technology, EPFL-LSL, Lausanne (Switzerland)
[2]Swiss Federal Institute of Technology, EPFL-LSA, Lausanne (Switzerland)

Abstract

We present SOS++, a bioinspired method combining evolution and learning, allowing the automatic design of the controller of autonomous agents, described as a finite-state machine. The application of this method to well-known problems, for example the follow-up of a trail or the resolution of a maze, led to the emergence of some behaviors we could qualify as intelligent. Moreover, it is possible to use the method in a hierarchical way in order to obtain complex behaviors starting from a set of basic actions. We have used an algorithm which is a variation of reinforcement learning with a reward adapted to the degree of uncertainty of the performed prediction.

Introduction

According to Brooks' work (Brooks 1999), the best way to program an autonomous robot is to provide it with a set of basic behaviors and to let more complex behaviors emerge by interactions between the robot and the environment.

The robot's creator must design the body, thus choosing a set of possible basic actions and perceptions: the actions are implemented by actuators (motors, for example), and the perceptions of the environment are acquired by sensors. The designer then implements some simple behaviors as a sequence of basic actions commanded by a controller, generally a neural network. More complex behaviors appear in an autonomous way, by learning and/or evolution of the controller while interacting with the environment. This same principle can be applied, in a more general way, to the design of autonomous agents.

In this paper we present another approach, using finite state machines (FSM) to implement the controller. As in the preceding case, the FSM controlling a complex behavior is the result of an autonomous process of interaction with the environment, using bioinspired methods of evolution and learning.

After a section explaining the bioinspired basis of our approach, we present a first method using a variation of reinforcement learning illustrated by a solution of the well-known Santa Fe trail problem (Koza 1992). Evolution is not always important for this first method (a solution for the Santa Fe trail is found during the first generation, for example) and, in fact, proves to be unable to provide solutions to more complex problems (the Los Altos Hills trail (Koza 1992), for example). In the next section, we present a second method, using two hierarchical levels of FSM: we added a set of FSM-implemented macroactions to the basic simple actions. This method finds a solution to the problem of Los Altos trail, but the designer has a more important contribution: the first hierarchical level of FSM, implementing the macroactions, is the result of her work and not the result of evolution or learning. We then present a third method, where evolution plays a greater role, that finds very interesting solutions for complex problems such as the Los Altos trail or the Wiering and Schmidhuber maze (Wiering & Schmidhuber 1997) using only the set of basic actions (e.g., our method is able to find sequences of actions that can be reused at different times, thus providing an automatic definition of macroactions and consequently of hierarchical behaviors). We finish with a concluding section and an outline of our future work, using real robots as examples.

Biological Inspiration

Evolution has provided each animal species with a set of basic actions which make it possible for its members to interact with their environment. From one species to another one can find similar actions but their implementation differs: evolution has found different solutions in each case. The execution of one of these actions, using several organs of the animal's body, is commanded by its nervous system (the brain for the more complex animals).

The realization of a given task implies the execution of several actions in a given sequence: it is a behavior. In this case, learning can play a greater role than evolution. The sequence of actions can be culturally transmitted to an individual during its life time, without need to wait several generations so that it gets genetically coded in its DNA (thus being transformed into an additional basic action).

This body-brain duality, (organs of the action)-

(command of the action), exists also in digital systems, as the pair (data path unit)-(control unit). In these cases, the data path unit contains all the elements of storage and data processing, while the control unit is a FSM responsible for sequencing the operations carried out in the data path unit.

In the case of an autonomous robot, the duality is clearer: the various sensors and actuators play the body role and its controller plays the brain role. The design of the robot's body is not left to evolution, as this would take an unbearably long time. Therefore, in the large majority of cases it is the task of an engineer: the design of the robot always includes the choice of its sensors and actuators. Although this task is far from being easy, it is a problem within a good engineer's reach. On the other hand, the design of the brain (the robot's controller) is much more difficult, particularly when one wants to see it carrying out complex,"intelligent", behaviors. For a living being, intelligence is the capacity to individually find the behavior to respond to an excitation of its environment or to solve a complex problem.

As with our analogy of living beings, it is common to use neural networks to implement the controller of autonomous agents. For these kind of networks, there are several well-known and tested learning algorithms, thus making the design easier: the designer chooses the starting network structure and the task is executed by self-organization, following a given learning algorithm. A more recent variation of this methodology uses genetic algorithms to evolve the neural network structure or some learning parameters (Nolfi & Floreano 2000).

Our approach is quite different, oriented towards a hardware implementation of the controller: the body of the agent, an autonomous robot for example, is designed by traditional methods, while we consider the controller as a FSM, commanding the possible actions of the body. A FSM is fully described by two elements: the set of its states and the transitions between them. In a Moore FSM, an action is associated to each state (Sanchez 1998). We use a genetic algorithm to find the set of states-actions pairs used by the learning algorithm. A variation of the reinforcement learning, a learning algorithm well known in the field of neural networks, is used to dynamically build the interconnections between the states. Some of the learning parameters are also calculated by the genetic algorithm.

To design a controller starting from its FSM description is a classic problem for digital systems engineers. In addition to the ease of implementation, this representation gives a much higher level of interpretability when compared to neural networks. Indeed it is much easier to explain and interpret a behavior starting from its FSM representation than of that of the neural network. The possibility to understand the way the solution works can help us when designing new systems or optimizing existing solutions.

First Method: SOS, A Variation of Reinforcement Learning

Reinforcement learning (Sutton & Barto 1998) is an iterative mathematical method rendering an agent able to learn how to carry out a given task while interacting with its environment by the means of rewards.

At each instant t, the agent receives a representation of the state of its environment, s^t. According to this information, the agent carries out an action selected among those which it is able to realize (this selection can be seen as a prediction). At the next moment, it receives a reward of value r^t and the environment passes to a new state s^{t+1}. To correctly carry out a given task, i.e. to correctly learn it, the agent must seek to maximize the sum of the obtained rewards: in other words, it must minimize the *temporal-difference error*, calculated as the difference between two predictions at successive moments.

For the choice of the action, the agent uses a *value function*, generally implemented as a *lookup table*, where each line corresponds to a state of the environment and each column to a possible action of the agent. Each entry in the table contains an *action value*, $Q(s, a)$, giving the maximum reward that an agent can hope to receive when, being at state s, uses action a. Learning implies correctly updating the values of the lookup table, as a function of the rewards obtained. This update is calculated using the following equations:

$$\delta = r^t + \gamma\, Q(s^{t+1}, a^{t+1}) - Q(s^t, a^t)$$

$$Q(s^t, a^t) \leftarrow Q(s^t, a^t) + \alpha\, \delta$$

where δ is the temporal-difference error, r^t is the reward obtained at time t, and α and γ are learning parameters.

With this method, the agent is able to correctly learn a task only if the state of the environment gives sufficient information: the state coding thus has a great influence on the learning quality. Our approach is completely different: we do not code the state of the environment but, quite simply, we use values from some of its parameters (those which are accessible to the input organs or sensors of the agent). These inputs of the agent (perceptions) will allow it to change its internal state: we thus learn the states of the agent's controller and not those of the environment. In this way, we use raw data from the environment, without needing to interpret them and to code them as states.

The controller of the agent is seen now as a FSM, where each state summarizes all the knowledge an agent needs to plan the continuation of its behavior. In addition, the transition between two states is done according to the perceived input. In the case of a Moore FSM, an action is associated to each possible state. Consequently,

	O_0				O_n		
	S_0	$\cdots$	S_m	$\cdots$	S_0	$\cdots$	S_m
S_0	0.5	0.3	0.6	$\cdots$	$T(S_0, O_n, S_i)$		
$\vdots$		$\vdots$		$\vdots$		$\vdots$	
S_m	$T(S_m, O_0, S_i)$			$\cdots$	$T(S_m, O_n, S_i)$		

Figure 1: Lookup table of estimations of the state values $T(S_m, O_n, S_i)$, where S_m is the current state of the environment, O_n is the current observation of the environment made by the agent, and S_i is the next state.

the relationship (state of the environment)-(action of the agent) of the traditional reinforcement learning now becomes a relationship (current state of the agent, input)-(next state of the agent). In other words, to learn now means to find the appropriate transition between two states of the agent for a given input.

FSMs are generally represented as state diagrams or, in an equivalent way, as lookup tables called state tables. We use a table inspired by state tables to replace the table of state-action values $Q(s, a)$ in our method. Figure 1 shows the format of such a table: there is a line per given state S_k and m columns (next states S_i) per input O_j. To each one of these triplets (an entry of the table) is associated a value $T(S_k, O_j, S_i)$, giving the maximum reward that the agent can hope to receive when, being in the state S_k, and perceiving the input O_j, it chooses S_i as the next state of S_k. Like in traditional reinforcement learning, to learn means to update the table. Thus, the equations for learning become in our case:

$$\delta = r + \gamma\, T(s', o', s'') - T(s, o, s')$$
$$T(s, o, s') \leftarrow T(s, o, s') + \alpha\, \delta \qquad (1)$$

where r, α and γ hold the same meaning as previously

The preceding equations perform the adaptation of a state-value function implemented by the $T(S_m, O_n, S_i)$ table. There is no action selection mechanism in our system. Instead, there is a single action associated to each state, as this is the case for a Moore FSM. Therefore, we must suppose that the state-action associations are the right ones. Indeed, we achieve this by using a genetic algorithm, as we will soon explain.

An individual (agent) of the population used by the genetic algorithm (Michalewicz 1996; Vose 1996) is represented by a fixed-size genome, containing as many genes as S_k states (note that the genome encodes a fixed maximum number of states, it is learning which further reduces this number). The value of a gene is an integer indicating the action associated to the corresponding state. For example, the genome 212 indicates that we work with a FSM containing a maximum of three states, and that state S_0 is associated to action 2, S_1 to action 1, and S_2 to action 2. At each generation, the less adapted $\frac{2}{3}$ of the population is replaced by new individuals resulting from the crossover of the remaining third (the elite, the fittest). The parents of these new individuals are selected by *roulette wheel*, and the offsprings may be modified by mutation. The crossover operation is done by cutting the two genomes in a single point, and always at genes' border. The mutation, carried out with a probability PMUT, randomly gives a new valid value to a gene. It is significant to notice that not all the genes of an individual are necessarily used in the solution of a problem: the learning will be given the responsibility to interconnect needed states, leaving the other states without connection.

To solve a given problem, we start by creating a random population. Then, the *fitness* of each individual is calculated as a function of its learning capabilities. This value is then used to determine the individuals authorized to reproduce. During its life, each individual thus uses the reinforcement learning method described previously, formalized in the following way, under the name of SOS *(State-Observation-State)* algorithm (Sanchez & Perez-Uribe & Mesot 2001):

1. Initialize the table. $\forall s \forall o \forall s' : T(s, o, s') = c$ where c is a constant equal to the maximum possible reward (optimistic initialization).

2. Place the agent at the initial position: define $s = S_0$ and observe the input o corresponding to the position.

3. Choose s' such that $T(s, o, s') = \max_{i \in [0, |S|[} T(s, o, S_i)$, where S is the set of the states. If several future states are possible, take that with the smallest index i.

4. Go to the state s', carry out the action corresponding to this state (given by the genome) and recover the reward r.

5. Observe the input o' corresponding to the new position and choose s'' such that $T(s', o', s'') = \max_{i \in [0, |S|[} T(s', o', S_i)$. Again, if several cases are possible, choose the smallest index i.

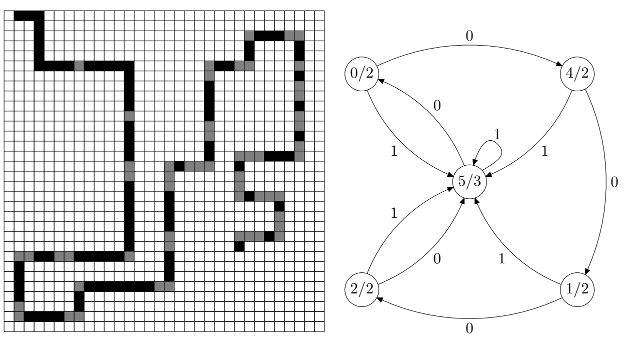
6. Update the value $T(s, o, s')$, using the equation:

$$\delta = r + \gamma\, T(s', o', s'') - T(s, o, s')$$
$$T(s, o, s') \leftarrow T(s, o, s') + \alpha\, \delta$$

7. Let $s = s'$, $s' = s''$, $o = o'$ and go to 4 as long as the end condition is not reached.

The Santa Fe Trail

The first problem on which we tested our algorithm is the Santa Fe trail, proposed by Christopher Langton (Koza 1992). An artificial ant is placed on a grid of 32 × 32 cells, of which some contain food (black cells in figure 2). The goal is to collect, in a limited number of steps, the greatest quantity of food, starting from the top left position. The ant "sees" only one cell in front of it: at every step, it receives an input of value 1 if there is food in the cell and 0 if not. At each step of the course, the ant can perform one of four possible actions, corresponding to the following genes: 0 (NOP), 1 (turn left), 2 (turn right) and 3 (move ahead a step). With each unit of collected food, the agent receives a positive reward ($r = 1$); in all the other cases, the reward is null ($r = 0$). The problem is to find a strategy enabling the ant to follow the trail in the most direct possible way, in spite of the trail's discontinuities.



(a) (b)

Figure 2: a) The Santa Fe trail and b) the best state diagram found by the SOS algorithm. The numbers on the arrows indicate the input value (0: no food, 1: food), the numbers on the circles indicate the state and the associated action respectively (0: NOP, 1: turn left, 2: turn right, 3: move ahead of a step).

We used our algorithm with the parameters given in the following table:

POP	SMAX	STEPS	TRIALS	PMUT
100	10	500	400	5%

where POP is the size of the population used, SMAX is the maximum number of states (i.e. the size of the genomes), STEPS determines the maximum number of actions which the agent is authorized to make in order to achieve the task, and TRIALS specifies the number of learning iterations (i.e. the number of times that the agent is placed back at the starting point and that a new chance is given to it to find a good strategy). Finally, PMUT indicates the probability of genome mutation. In addition, the learning parameters use the following values: $\alpha = 0.6$ and $\gamma = 0.9$. For the evolution, the measurement of fitness more strongly supports the individuals who accumulate maximum units of food, rather than those which have a minimum of states, and, finally, those which use a minimum of actions. Therefore we compute the fitness according to the following equation:

$$f = w_1 a_{\mathrm{rew}} + w_2(|S| - a_{\mathrm{st}}) + w_3(STEPS - a_{\mathrm{stp}}) \quad (2)$$

where a_{rew} is the sum of all rewards obtained by the agent, a_{st} is the number of used states and a_{stp} is the number of steps needed to go all over the trail. These values are those computed during the last trial of the agent on the trail. For this problem, the weights w_i are set as follows: $w_1 = 10^5$, $w_2 = 10^3$ and $w_3 = 1$. Thus individuals with good fitness will tend to collect as many foods pellets as possible, using the smallest number of states and steps.

In less than 10 generations, our algorithm is always able to find a solution requiring only 5 states, after about 30 learning iterations. One of the minimal solutions very often obtained is presented in figure 2. By taking a glance at this solution, some remarks can be made. Firstly, the evolution action can be regarded as unimportant: indeed, good solutions are always obtained after less than 10 generations and, in certain cases, even at generation 0. This implies that we are dealing with an easy problem since the learning can use virtually any genome in order to obtain a solution, provided that it contains at least two actions. Secondly, only two actions out of four are used: turn right (2) and move ahead of a step (3). Last but not least, the FSM representation renders the solutions highly understandable. Indeed, the analysis of the state diagram of figure 2 enables us to easily understand the behavior of the agent: each time the ant perceives an input of value 1, it passes to state 5, where it carries out the action 3 (move ahead of a step). Not perceiving food forces it into a cycle of exploration, turning each time to the right (action 2) to seek food. If food is not found after a full rotation, the ant moves straight ahead. In short, trail discontinuities are always in straight lines with food and their end can be determined by a simple rotation of the ant.

Second Method: SOS+, Learning with Hierarchical Actions

The simplicity of the Santa Fe trail led us to test another problem: the Los Altos Hills trail, proposed by Koza (Koza 1992). This trail is in fact an extension of the Santa Fe trail, since eight deviations were added there in its final portion (see figure 3). Among these additions, four are deviations no longer presenting any direction guide (as in the Santa Fe case): inside a discontinuity, the agent cannot determine the direction to follow simply by turning on the spot.

Los Altos Hills

Initially, our SOS algorithm did not find a solution to the problem, even by using great values for parameters

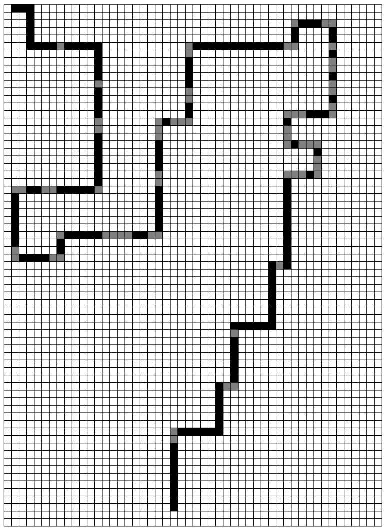


Figure 3: The Los Altos Hills trail

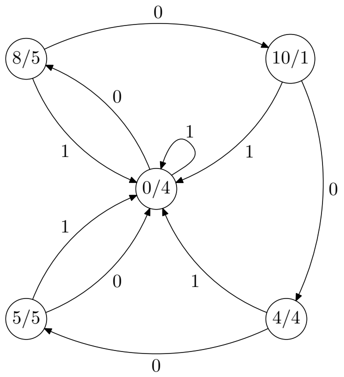


Figure 4: The best state diagram found by the SOS+ algorithm for Los Altos trail. Actions 4 and 5 are macroactions.

STEPS and TRIALS (2000 and 1000 respectively): no agent was able to collect the totality of the 157 units of food. This observation led us to suppose that this problem is too complex to be entirely solved with the basic actions NOP (0), turn left (1), turn right (2) and move ahead of a step (3). This is the reason for which we brought two new actions (in fact, Koza had also been obliged to add other actions for his solution). Our approach is hierarchical: the two new actions, called macroactions, are in fact other FSMs using the four basic actions. The two following paragraphs give a description of the behavior associated with each of the two macroactions we added.

Action 4 With this action, the agent is able to move ahead two steps, to turn on itself, then to return to its initial position but in the direction opposed to that which it had at its departure. If during this action a unit of food is found, the macroaction stops and the machine goes to the next state corresponding to an input equal to 1. If the macroaction comes to an end, the next state is selected according to the value of the input at this time. The sequence of the actions corresponding to this macroaction is as follows: 3 3 1 1 1 2 3 3.

Action 5 This action is similar to the preceding action, but with an intermediate exploration: after a step ahead, the agent turns on itself "to observe" if at this stage there is not already a unit of food showing the way to be followed; if this is not the case, it still moves ahead a step, then turns on itself and finally returns to its initial position, but in the direction opposed to that which it had at its departure. The principle of connection and end of this action is the same one as that of action 4. The sequence of the actions carried out is as follows: 3 1 1 1 1 3 1 1 1 2 3 3.

The parameters of the algorithm used for this problem are given in the following table:

POP	SMAX	STEPS	TRIALS	PMUT
500	15	10'000	500	5%

The values of the learning parameters α and γ are still 0.6 and 0.9, respectively. The fitness is computed using the equation 2, where the weights w_i are set as follows: $w_1 = 10^7$, $w_2 = 10^5$ and $w_3 = 1$.

After less than 10 generations, and less than 100 learning iterations, the SOS+ algorithm finds an agent which collects all 157 units of food in 963 actions, by using 5 states out of the 15 available in the genome. Figure 4 shows one of the solutions obtained: it required only 17 learning iterations, and 8 generations.

As expected, this solution is very similar to that of Santa Fe, with a more thorough exploration obtained thanks to the use of macroactions 4 and 5. If the number of states seems very small (5), it should not be forgotten that macroaction 4 breaks up in fact in 8 simple states and macroaction 5 in 12: indeed, at the lowest level of hierarchy, the number of states is equal to 41 (i.e., taking into account only the simple actions used in our first method).

Third Method: SOS++, Adaptive Rewards plus Evolution of Learning Parameters

Although the macroactions previously described make it possible to solve the problem of Los Altos Hills, this approach presents several disadvantages. First of all, the macroactions must be defined by the designer, who thus takes a very significant part in the resolution of the problem. Then, their presence in the solution can imply a significant increase in the number of basic states, and the understanding capability gained by their addition is obtained with the detriment of this number.

These disadvantages led us to more thoroughly analyze the failure of the first version of algorithm SOS, and therefore try to improve our method once more. A first reason to this failure comes from "over-learning": indeed, as more of the first half of Los Altos Hills problem is identical to the Santa Fe trail, our method quickly obtains an agent able to efficiently traverse this part by receiving a very great reward. It then becomes very difficult to make the agent change its strategy for the second part. The following subsection presents a solution to this problem.

Interpreting α

To control this "over-learning" problem, our first approach was to analyze the role played by the two learning parameters of the SOS algorithm, that of α more particularly.

Rewriting equation 1 as:

$$T(s,o,s') \leftarrow (1-\alpha)\,T(s,o,s') + \alpha\,[T(s,o,s') + \delta]$$

allows us to note that the updated value of $T(s,o,s')$ is composed by the sum of two expressions. The first, $(1-\alpha)\,T(s,o,s')$, is the current value of the table for the transition (s,o,s'), weighted by a factor $(1-\alpha)$. The second, $\alpha\,[T(s,o,s')+\delta]$, represents the value corrected by the learning for the same transition, weighted by a factor α. In other words, α indicates the percentage that the corrected value represents in the update of $T(s,o,s')$. Consequently, it becomes obvious that the value of α very strongly determines the learning quality and that it must be therefore adjusted according to the problem to solve. This is why it appeared necessary to us to include α and γ in the genome of an individual. The genome of an agent will be thus composed by a certain number of integer values (the actions associated with the states) and by two floating point values (the two parameters α and γ).

Island Evolution

Another modification was introduced in the algorithm in order to be able to process genomes of different size: the total population is now distributed in several islands, the inhabitants of each island having all the same size of genome, different from that of the other islands. Regularly, after a given number of generations, a given number of the best individuals of an island emigrate towards another island, thus exchanging their advantages. The sizes of the migrant genomes must adapt to that of the destination island: if it is smaller, the genomes are cut; in the other case, random genes are added. In all the cases, all the individuals of an island preserve the same size.

Adaptive Reward

In the first version of our SOS algorithm, the exploration of new solutions is limited by way of choosing the next state when several cases are possible, i.e. when all cases present the same maximum value of $T(s,o,s')$. In this case, one selects the state s' that presents the smallest index as the next state, and the other states will never be visited (unless the corresponding values change, of course).

In order to succeed with this limitation, we assign to each possible transition a uniform probability to be used: if there are n valid transitions, each one is assigned a probability of $\frac{1}{n}$. This probability will be employed to modify the reward granted to the agent after the choice of the next state. A negative reward (punishment) must be less significant if at the time of the choice of the transition there were several possibilities than if there was only one of them (i.e., a certainty should be punished more than a doubt, when it proves to be incorrect). In an equivalent way, a good choice of transition is rewarded more if there were several possibilities than if there was only one of them.

To quantify this uncertainty, we use the Shannon entropic measurement (Shannon 1948):

$$H = -\sum_{i=1}^{n} p_i \cdot \log(p_i)$$

where p_i is the probability of the ith choice.

As in our case the choices are equiprobable, one has $p_i = p = \frac{1}{n}$ where n is the number of choices. This leads to a simpler formula for the entropy:

$$H = -\log\left(\frac{1}{n}\right) = \log(n)$$

According to the assumptions formulated previously, the reward r must be proportionally weighted with the entropy if $r > 0$, and conversely if $r < 0$. To model this fact, we use weighting $\theta(n)$, defined by:

$$\theta(n) = \begin{cases} 1+\log(n) & \text{if } r > 0 \\ 1+\log(1+n_{\max}-n) & \text{if } r < 0 \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

where $n_{\max}$ is the maximum number of possible choices, i.e. the maximum number of states.

The final version of the learning SOS algorithm, called SOS++, is now:

1. Initialize the table. $\forall s \forall o \forall s' : T(s,o,s') = c$ where c is a constant equal to the maximum possible reward (optimistic initialization).

2. Place the agent at the initial position: define $s = S_0$, $r = 0$ and observe the input o corresponding to the position.

3. Choose s' such that $T(s,o,s') = \max_{i\in[0,|S|[} T(s,o,S_i)$, where S is the set of states; if several states are possible, choose one of them randomly.
4. Calculate $\theta_1 = \theta(n)$ using equation 3, where n is the number of possible choices at the preceding stage.
5. Go to the state s', apply the action associated to this state (given by the genome) and recover the reward r.
6. Observe the input o' corresponding to the new position and choose s'' such that $T(s',o',s'') = \max_{i\in[0,|S|[} T(s',o',S_i)$; again, if several cases are possible, choose one of them randomly.
7. Calculate $\theta_2 = \theta(n)$ using equation 3, where n is the number of possible choices at the preceding stage.
8. Update the value $T(s,o,s')$ using the equation:

$$\delta = \theta_1\, r + \gamma\, T(s',o',s'') - T(s,o,s')$$
$$T(s,o,s') \leftarrow T(s,o,s') + \alpha\,\delta$$

9. Let $s = s'$, $s' = s''$, $o = o'$, $\theta_1 = \theta_2$ and go to 5 as long as the end condition is not reached.

Los Altos Hills

Two tests were carried out on the problem of Los Altos Hills, using the islands model in both cases for the evolution and evolving the learning parameters. However, in the first test, the agent used our first SOS algorithm for learning, whereas in the second test, SOS algorithm with adaptive reward was used. For the two cases the agent has only the 4 basic actions: NOP (0), turn left (1), turn right (2) and move ahead of a step (3).

For the first test, the values of the parameters are as follows:

NISL	ISLPOP	NGBM	NEMIG
9	100	5	3

STEPS	TRIALS	PMUT	PREP
2000	1000	5%	75%

In the first table, NISL is the number of islands, ISLPOP is the number of individuals per island, NGBM is the number of generations before migration, and NEMIG is the number of emigrants. In the second table, STEPS, TRIALS and PMUT have the same meaning as before and PREP gives the percentage of individuals replaced at each generation.

For this test, the islands were populated with genomes of size 40, 50, 60, 70, 80, 90, 100, 30, and 20 (one size per island). The fitness of each individual is computed using the equation 2, where the weights w_i are set as follows: $w_1 = 10^7$, $w_2 = 10^4$ and $w_3 = 1$. The best solution was obtained on island number 6, after 6 generations. It uses only 12 of the 100 available states, making the course of the entire trail in 999 actions and using only 283 learning iterations of the 1000 available. The end values of the learning parameters for this solution were: $\alpha \approx 0.38$ and $\gamma \approx 0.98$. The state diagram obtained is given in figure 5.

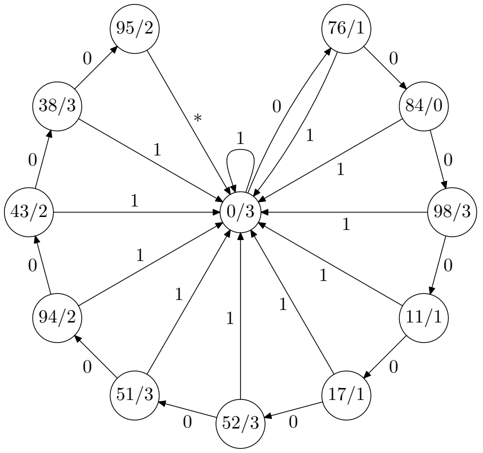


Figure 5: The best state diagram found by the SOS++ algorithm for Los Altos trail, without adaptive reward and with the NOP action.

An analysis of this graph enables us to understand the behavior of the agent. It begins with state 0 (move ahead of a step) and it remains there as long as it perceives an input 1; as soon as it is no longer the case, it enters in an exploration cycle starting at state 76. This cycle leads it to carry out the actions sequence 1 0 3 1 1 3 3 2 2 3 2, which corresponds to explore a cell on the left (1 0 3 1 1 3), then a cell on the right (3 2 2 3), then to return to the initial orientation (2). This behavior is very similar to the macroactions which we previously defined for the same problem, except that here it is found by learning. If, during this cycle, the agent perceives an input 1, it returns to state 0 to move straight ahead, until next discontinuity. It is interesting to note the presence of an action NOP (even if this action has not been useful in our experiments, we left it to respect the original specification of the problem. Interestingly, it had always been eliminated by the learning and evolution algorithm). In fact, for learning, action NOP is as valid as the actions turn left or turn right, since all three generate a null reward; it is thus the evolution's responsibility to select better individuals, i.e. individuals not including in its genome the action NOP. However, an individual collecting all of the 157 units of food with only 12 states, as it is the case in our solution, has an enormous advantage over its colleagues. With such a large advantage, it is unlikely to be dethroned before many generations.

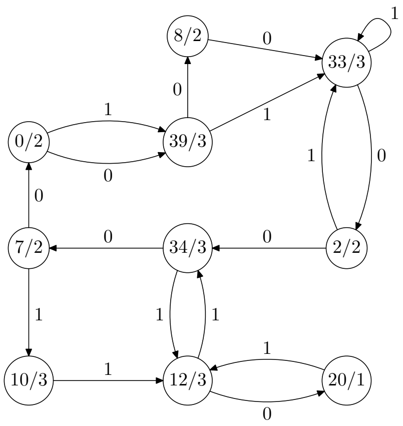


Figure 6: The best state diagram found by the full SOS++ algorithm for Los Altos trail, without the NOP action.

For the second test we removed the state NOP. The parameters used were the same as those of the first test, but with 10 islands populated of genomes of size 100, 90, 80, 70, 60, 50, 40, 30, 20, and 15 (one size per island). After only 3 generations, on island number 6, an individual was found able to traverse the trail in 679 actions, using only 10 states out of the 40 of its genome. This solution, which required 472 learning iterations, is given in figure 6.

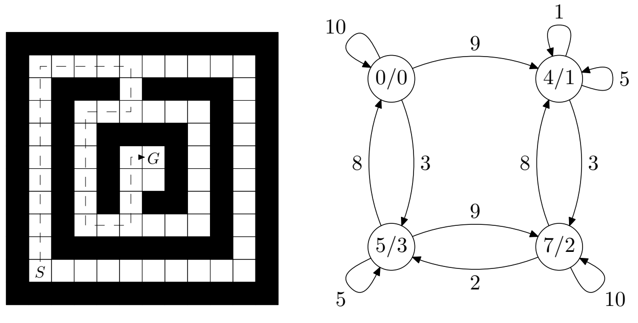
In spite of the low number of states used, this solution remains nevertheless interpretable and presents a behavior quite surprising by its "intelligence". Indeed, the state diagram of figure 6 can be broken up into two sub-FSM; the first, composed of the states 0, 39, 8, 33, 2, 34, 7, and the second of the states 34, 7, 10, 12, 20, which corresponds to the two parts of the trail: that identical to the Santa Fe trail and that added later by Koza. Our method was thus able to differentiate between these two parts, finding a solution for each one and interconnecting them. For the first part, the agent leaves state 2 and carries out the sequence of actions 2 3 2 2 3 (2) 3: when it is facing a discontinuity, it explores a cell to the right (2 3 2 2 3) and, if its still does not perceive anything (input 0), turns right (2) and moves straight ahead (3). This behavior makes it possible to navigate the complete Santa Fe trail. To change behavior, the agent must find the breaking point with Santa Fe, i.e. where the trail added by Koza starts. The strength of this method is shown when, in a surprising manner, this characteristic is actually found at the point of the first contour added by Koza. When it arrives there, the agent moves towards the south and is in state 33, with an input 0. This configuration leads it to turn right (state 2), then to move ahead (state 34). At this point, the agent is directed towards the west and perceives an input 1, which enables it to know that it engages in the second part of the trail (state 12). Here, it moves ahead, then turns left, so as to move towards the south, and oscillates between the states 12 and 34, which both make it move ahead. When the agent arrives at the following contour, it is at state 34 and perceives an input 0, turns right (state 7), then passes successively through states 10, 12 and 34 which make it move towards the next discontinuity. When it reaches that point, it is at state 34 with an input 0 and comes back to the first sub-FSM, which leads it to move backwards (actions 2 2 3), going back a cell (2 3) and turning to the west (2 3). At the end of this set of actions, the agent is at a cell below the discontinuity (on the gray cell). As input 0 is perceived on this site, this leads the agent to perform this set of actions for a second time, enabling it to find the trail, i.e. to have an input 1 when it is at state 2. A passage through state 33, then through state 2, enables it to be directed towards the south; then, the return to state 33 enables it to reach the next difficulty. The latter is passed in the same way as the preceding one, i.e. by exploring a cell on the right while going down, but this time the agent is at state 34 when it finds the trail. The final part of the trail is carried out thanks to the second sub-FSM. The last difficulty is then overcome by using the same exploratory behavior used in the previous discontinuity.

Wiering and Schmidhuber Maze

Reinforcement learning works particularly well for the trail problems because a reward is possible at almost all the steps of the problem. It is much more difficult to use this learning method if the reward is given only at the end, when the task is successfully accomplished. This is the case of the Wiering and Schmidhuber maze (Wiering & Schmidhuber 1997) (figure 7), a well-known problem in literature, for which traditional reinforcement learning does not give a solution.

Here, starting at the point S, the task of the agent is to arrive at the point G while following the shortest possible way. To accomplish this task, it can move in the four cardinal directions, with actions coded in the following way: 0 (west), 1 (north), 2 (east) and 3 (south). The knowledge of its environment comes to it from 4 sensors, placed in the 4 directions: the value of the input (observation) is thus the sum of the 4 sensors, weighted by 1 (west), 2 (north), 4 (east) and 8 (south). Thus, for example, walls in the west and the south of the agent generate an input $1 + 8 = 9$. In addition, the agent obtains a reward of -1 ($r = -1$) when it runs up against a wall, of 10 when it achieves the goal, and of 0 otherwise. In the evolution, the fitness rewards individuals reaching to the goal more than those using fewer operations to do so, and finally those using fewer states. Therefore, using the equation 2, we set the weights w_i as follows: $w_1 = 10^5$, $w_2 = 1$ and $w_3 = 10^2$.

The evolution and learning parameters used in our tests are as follows:



(a) (b)

Figure 7: a) Wiering-Schmidhuber maze and b) the best state diagram found by the SOS++ algorithm. The numbers on the arrows indicate the input value computed as a function of its four sensors (e.g., 0: no obstacle, 1: obstacle to the right, 2: obstacle to the north, 6: obstacle to the north and to the west, etc.), the numbers on the circles indicate the state and the associated action (0: west, 1: north, 2: east, 3: south).

NISL	ISLPOP	NGBM	NEMIG
9	200	5	3

STEPS	TRIALS	PMUT	PREP
250	1000	5%	75%

The evolution uses nine islands, populated by genomes of size 20, 15, 10, 9, 8, 7, 6, 5, and 4 (one size per island). After 548 generations, our algorithm found an agent passing through the shortest way (28 actions) and using only 4 states among the 8 available in its genome, after 662 learning iterations, with $\alpha \approx 0.24$ and $\gamma \approx 0.93$. The state diagram corresponding to this solution is given in figure 7. It is interesting to note that although the optimal solution required 548 generations, other solutions are also obtained in the first generation, only by learning. These latter solutions generally use the shortest way, but none have less than 7 states. This result is very significant since, as told previously, the traditional reinforcement learning never solves this problem.

Conclusions

The use of FSMs to realize the controller of autonomous agents is preferable to the use of neural networks, when planning to produce a hardware implementation: indeed, this method is the most commonly used when designing digital systems. Moreover, FSMs have the advantage of their “legibility”: it is easier to understand a behavior if it is described in this form than as a neural network, for example.

Our method allows the automatic synthesis of the behaviors which an autonomous agent can perform with this type of controllers. These behaviors emerge by evolution and learning from the simple list of basic actions with which the agent was equipped at its “birth”, and from the interactions between these actions and the environment. In addition, we can generalize the found solutions: the FSM produced by the solution of the Sante Fe problem, for example, can be used to correctly navigate through a whole family of other trails without additional learning (those where the trail discontinuities are always in straight lines with food and their end can be determined by a simple rotation of the ant).

Evolution, used in a traditional way, allows one to find good individuals who are able to achieve the expected behavior simply by learning. A very fast evolution is achieved thanks to the very small size of the genome, made up simply of a list of actions associated with the possible states of the FSM and two learning parameters.

Our learning method, derived from classical reinforcement learning, modified by the use of an evolution of its two learning parameters together with a reward adapted to the uncertainty of the prediction performed, proved very powerful. Indeed, learning can solve simple problems without evolution, finding a solution at the first generation. In addition, problems for which traditional reinforcement learning could not obtain any solution were solved.

The discrete character of the values used for the observations of the environment could be regarded as a disadvantage if one wants to use this method with real problems, such as the control of autonomous robots. However, it is possible to consider our basic actions as FSMs processing the lowest level of perception, and our method can deal in these cases with higher level behaviors. In fact, we are currently implementing the experiments described in this paper with a real Khepera robot: a film with these new results is viewable on the Web page of our Laboratory (`http://lslwww.epfl.ch`).

References

Brooks, R. A. 1999. *Cambrian Intelligence: The Early History of the New AI.* Cambridge, MA: MIT Press.

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

Michalewicz, Z. 1997 *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Heidelberg, third edition.

Nolfi, S., and Floreano, D. 2000. *Evolutionary Robotics.* Cambridge, MA: MIT Press.

Sanchez, E. 1998. An Introduction to Digital Systems. In Mange, D., and Tomassini, M., eds., *Bio-Inspired Computing Machines: Toward Novel Computational Machines*, 13. Lausanne: Presses Polytechniques et

Universitaires Romandes.

Sanchez, E., Perez-Uribe, A., and Mesot, B. 2001. Solving Partially Observable Problems by Evolution and Learning of Finite-State Machines. In Liu, Y., Tanaka, K., Iwata, M., Higuchi, T., and Yasunaga, M., eds., *Evolvable Systems: From Biology to Hardware, ICES 2001*, 267. Berlin: Springer-Verlag.

Shannon, C. E. 1948. A Mathematical Theory of Communication. *Bell Systems Technical Journal* 27:379.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press.

Vose, M. D. 1999 *The Simple Genetic Algorithm.* MIT Press, Cambridge, MA.

Wiering, M., and Schmidhuber, J. 1997. HQ-Learning. *Adaptive Behavior* 6:2.