TRAINING AGENTS TO PERFORM SEQUENTIAL BEHAVIOR*

Marco Colombetti+ Marco Dorigo#

TR-93-023

September 1993

Abstract

This paper is concerned with training an agent to perform sequential behavior. In previous work we have been applying reinforcement learning techniques to control a reactive robot. Obviously, a pure reactive system is limited in the kind of interactions it can learn. In particular, it can only learn what we call pseudo-sequences, that is sequences of actions in which the transition signal is generated by the appearance of a sensorial stimulus. We discuss the difference between pseudo-sequences and proper sequences, and the implication that these differences have on training procedures. A result of our research is that, in case of proper sequences, for learning to be successful the agent must have some kind of memory; moreover it is often necessary to let the trainer and the learner communicate. We study therefore the influence of communication on the learning process. First we consider trainer-to-learner communication introducing the concept of reinforcement sensor, which let the learning robot explicitly know whether the last reinforcement was a reward or a punishment; we also show how the use of this sensor induces the creation of a set of error recovery rules. Then we introduce learner-to-trainer communication, which is used to disambiguate indeterminate training situations, that is situations in which observation alone of the learner behavior does not provide the trainer with enough information to decide if the learner is performing a right or a wrong move. All the design choices we make are discussed and compared by means of experiments in a simulated world.

---

* This work has been partly supported by the Italian National Research Council, under the "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo", subproject 2 "Processori dedicati", and under the "Progetto Finalizzato Robotica", subproject 2 "Tema: ALPI".

+ Progetto di Intelligenza Artificiale e Robotica, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milano, Italy (e-mail: colombet@ipmel2.elet.polimi.it).

International Computer Science Institute, Berkeley, CA 94704, and Progetto di Intelligenza Artificiale e Robotica, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milano, Italy (e-mail: dorigo@icsi.berkeley.edu).

1. Introduction

This paper is concerned with the application of evolutionary reinforcement learning to the development of agents that act in a given environment.

Machine learning techniques have been widely adopted to shape the behavior of autonomous agents in partially unpredictable environments. Most often, agents are viewed as *reactive systems*, that is as systems whose actions are completely determined by current sensorial input. Several works in the literature, both theoretical and experimental, show that reactive systems can learn to carry out fairly complex tasks (see for example Mahadevan & Connell, 1992; Dorigo & Colombetti, 1992); however, there are interesting behavioral patterns that just cannot be exhibited by reactive systems, in that they are not determined by current perceptions alone.

An important class of non-reactive tasks is the class of *sequential behavior patterns*, that is behaviors in which the decision of what action to perform at time t is influenced by the actions performed in the past. The problem of learning sequential behavior has been tackled by Singh (1992) in the context of Q-learning. In this paper, we present a different approach to the problem of learning sequential behavior patterns, viewed as the result of coordinating separately learned basic behaviors (Colombetti & Dorigo, 1992).

The work presented here is part of a wider research effort aimed at developing agents capable of complex behavior through both explicit design and machine learning. In our research, which has a strong experimental orientation, we use ALECSYS, a software tool designed by Dorigo (1992). ALECSYS allows one to implement an agent as a network of interconnected modules, each of which is a learning classifier system (Booker, Goldberg & Holland, 1989). The system, which runs in parallel on a network of transputers, has been connected to both simulated agents and physical robots. The behavior of agents implemented through ALECSYS is shaped through a supervised reinforcement scheme, that is through reinforcements provided by an external trainer observing the agent's behavior.

Our general methodology is the following. First, we define an environment, an agent, and a target behavior pattern that we want the agent to exhibit in the environment. Then we design a sensorimotor interface and a modular control architecture for the agent; typically, we use a hierarchical architecture where lower level modules are in charge of implementing basic reactive responses, and higher level modules are in charge of coordinating such responses to execute the overall task. We also design a training policy, i.e. a strategy to train the agent, and implement the trainer as a computer program in charge of giving reinforcements to the agent. Finally, we plan and execute a number of experiments to see whether the target behavior emerges, and to analyze the effect of different design choices on the agent's performance.

In this paper we present the results of a research aimed at developing sequential behavior in a simulated agent. In particular, we concentrate on the problem of coordinating previously learned basic behaviors in such a way that a sequential behavior pattern will emerge. In Section 2, we define some technical terminology and situate the problem of sequential behavior in the context of a general approach to the development of autonomous agents. Section 3 briefly describes ALECSYS. In Section 4 we define the agents, the environment and the behavior patterns on which we have carried out our experimentation, and specify our experimental plan. In Section 5 we describe the two different training strategies which we used in our experiments. In Section 6, we report the results of a number of simple experiments. Concluding remarks are given in Section 7.

2. Reactive and dynamic behavior

As we have already pointed out, the simplest class of agents is that of *reactive systems*, i.e. agents which react to their current perceptions (Wilson, 1990; Littman, 1992). In a reactive system, the action a(t) produced at time t is a function of the sensorial input s(t) at time t:

$$a(t) = f(s(t)).$$

As argued by Whitehead & Lin (1993), reactive systems are perfectly adequate to *Markov environments*, i.e. when:

- The *greedy control strategy* is globally optimal. This means that choosing the locally optimal action in each environmental situation leads to a course of actions that is globally optimal.
- The agent has *complete knowledge* of both the effects and the costs (or gains) of each possible action in each possible environmental situation.

In this case, there are well-known learning schemes, like Q-learning (Watkins, 1989; 1992), that are demonstrably able to discover the optimal control strategy through experience. In other words, a learning reactive agent can improve its performance so that it asymptotically converges to the optimal control strategy.

Although fairly complex behaviors can be carried out in Markov environments, very often an agent cannot be assumed to have complete knowledge about the effects and/or the costs of its own actions. Non-Markov situations are basically of two different types:

- *Hidden-state environments*. A *hidden state* is a part of the environmental situation that is not accessible to the agent, but is relevant to the effects and/or to the costs of actions. If the environment includes hidden states, a reactive agent cannot choose an optimal action; for example, a reactive agent cannot decide an optimal movement to reach an object that it does not see.

- *Sequential behavior*. Suppose that at time t an agent has to choose an action as a function of the action performed at time $t-1$. A reactive agent can perform an optimal choice only if the action performed at time $t-1$ has some characteristic and observable effect at time t, that is only if the agent can infer which action it performed at time $t-1$ by inspecting the environment at time t. For example, suppose that the agent has to put an object in a given position, and then to remove the object. If the agent is able to perceive that the object is in the given position, it will be able to appropriately sequence the placing and the removing actions. However, a reactive agent will be unable to act properly at time t if:
  - the effects of the action performed at time $t-1$ cannot be perceived by the agent at time t (this is a subcase of the hidden-state problem); or:
  - no effect of the action performed at time $t-1$ persists in the environment at time t.

To develop an agent able to deal with non-Markov environments, one must go beyond the simple reactive model. We say that an agent is *dynamic* if the action $a(t)$ it performs at time t depends not only on its current sensorial input $s(t)$, but also on its *state* $x(t)$ at time t; in turn, such state and the current sensorial input determine the state at time $t+1$[1]:

$$a(t) = f(s(t),x(t)); \quad (1)$$
$$x(t+1) = g(s(t),x(t)).$$

In this way, the current action can depend on the past history.

Agent's states can be called *internal states*, to distinguish them from the states of the environment. They are often regarded as memories of the agent's past, or as representations of the environment; however, in spite of their rather intuitive meaning, terms like *memory* or *representation* can easily be used in a confusing way. Take for example the packing task proposed by Lin & Mitchell (1992) as an example of non-Markov problem:

> "Consider a packing task which involves 4 steps: open a box, put a gift into it, close it, and seal it. An agent driven only by its current visual percepts cannot accomplish this task, because when facing a closed box the agent does not know if the gift is already in the box and therefore cannot decide whether to seal or open the box."

It seems that the agent needs to remember that it has already put a gift into the box. In fact, the agent must be able to assume one of two distinct internal states, say 0 and 1, so that its controller can choose different actions when the agent is facing a closed box. We can associate state 0 to

---

[1] In automata theory, this definition corresponds to a class of automata known as *Mealy machines* (McCluskey, 1986).

“the box is empty”, and state 1 to “the gift is in the box”. Clearly, the state must switch from 0 to 1 when the agent puts the gift into the box. But now, the agent's state can be regarded:

- as a *memory* of the past action “put the gift into the box;” or:
- as a *representation* of the hidden environmental state “the gift is in the box.”

Probably, the choice of one of these views is a matter of personal taste. But consider a different problem. There are two distinct objects in the environment, say A and B. The agent has to reach A, touch it, then reach B, touch it, then reach A again, touch it, and so on. In this case, provided that touching an object does not leave any sign on it, there is no hidden state in the environment to discriminate the situations in which the agent should reach A from the ones in which it should reach B. We say that the environment is *forgetful*, in that it does not keep track of the past actions of the agent. Again, the agent must be able to assume two distinct internal states, 0 and 1, so that its task is to reach A when in state 0, and to reach B when in state 1. Such internal states cannot be viewed as representations of hidden environmental states, because these do not exist. However, we still have two possible interpretations:

- internal states are *memories* of past actions: 0 means that B has been touched, and 1 means that A has been touched;
- internal states are *goals*, determining the agent's current task: state 0 means that the current task is to reach A, state 1 means that the current task is to reach B.

The conclusion we draw is that terms like *memory*, *representation* and *goal*, which are very commonly used for example in artificial intelligence, often involve a subjective interpretation of what is going on in an artificial agent. The term *internal state*, borrowed from systems theory, seems to be neutral in this respect, and it describes more faithfully what is actually going on in the agent.

In this paper, we are concerned with internal states that keep track of past actions, so that the agent's behavior can follow a sequential pattern. In particular, we are interested in dynamic agents possessing internal states by design, and learning to use them to produce sequential behavior. The idea is that the dynamics of the agent’s state will act as a kind of action plan, able to enforce the correct sequencing of actions. However, this intuitive idea must be taken with some care.

Let us observe that not all behavior patterns that at first sight appear to be the based on an action plan are necessarily dynamic. Consider for example an example of *hoarding behavior*: an agent leaves its nest, chases and grasps a prey, brings it to its nest, goes out for a new prey, etc. This apparently dynamic behavior can be produced by a reactive system, whose stimulus-

response associations are described by the following production rules (where only the most specific production whose conditions are satisfied is assumed to fire at each cycle):

Rule 1:				→	*move randomly*.
Rule 2:	*not grasped*	&	*prey ahead*	→	*move ahead*.
Rule 3:	*not grasped*	&	*prey at contact*	→	*grasp*.
Rule 4:	*grasped*	&	*nest ahead*	→	*move ahead*.
Rule 5:	*grasped*	&	*in nest*	→	*drop*.

In fact, we ran several experiments showing that a reactive agent implemented with ALECSYS can easily learn to perform similar tasks.

It is interesting to see why the behavior pattern described above, while merely reactive, appears as sequential to an external observer. In fact, if instead of the agent's behavior we consider the behavior of the global dynamic system constituted by *the agent and the environment*, the task is actually dynamic. The relevant states are the states of the environment, which keeps track of the effects of the agent's moves; for example, the effects of a grasping action are stored by the environment in the form of a grasped prey, which can then be perceived by the agent. In the following, we shall call *pseudo-sequences* those tasks performed by a reactive agent that are sequential in virtue of the dynamic nature of the environment, and we shall reserve the term *proper sequence* for tasks that can be executed only by dynamic agents, in virtue of their internal states.

Let us rephrase the above considerations. As it has already been suggested in the literature (see for example Rosenschein & Kaelbling, 1986; Beer, 1993), the agent and the environment can be viewed as a global system, made up by two coupled subsystems. For the interactions of the two subsystems to be sequential, at least one of them must be properly dynamic, in the sense that its actions depend on the subsystem's state. The two subsystems are not equivalent, however, because while the agent can be shaped to produce a given target behavior, the dynamics of the environment is taken as given, and cannot be trained. It is therefore interesting to see whether the only subsystem that can be trained, that is the agent, can contribute to a sequential interaction with states of its own: this is what we called a proper sequence.

In this paper, instead of experimenting with sequences of single actions, we have focused on tasks made up of a sequence of *phases*, where a phase is a subtask which may involve an arbitrary number of single actions. Again, the problem to be solved is: how can we train an agent to switch from the current phase to the next one on the basis of both the current sensory input and knowledge of the current phase?

One important thing to be decided is when the phase transition should occur. The most obvious assumption is that a *transition signal* is produced by the trainer or by the environment, and is perceived by the agent. Clearly, if we want to experiment on the agent's capacity to produce proper behavioral sequences, the transition signal must not itself convey information about which should be the next phase.

3. The learning system

ALECSYS is a tool to develop learning agents. Using ALECSYS, the learning "brain" of an agent can be designed as the composition of many learning behavioral modules. Some of these modules, which we call *basic behaviors*, are directly connected with sensorial and actuatorial routines, and their task is to learn responses to external stimuli. In some architectural organizations, namely in hierarchical architectures, we define a second kind of modules, called *coordination behaviors*, whose main task is to learn to coordinate other behaviors. Coordination modules are connected to lower-level modules, both basic and coordination ones, and can either choose which of the actions proposed by connected modules should be given priority, or compose such actions into a complex behavioral response.

There are different ways in which behavioral modules can be put together to build a learning system. In a recent study we have investigated a number of them (Dorigo & Colombetti, 1992), which we briefly summarize below.

- *Monolithic architecture*. In this architecture (see Figure 1) there is only one learning module which is in charge of learning all the behaviors necessary to accomplish the task. It is the most straightforward way of building a learning system, but it is not very efficient. In fact, it was the inefficiency of this approach which first motivated *distributed* architectures.
- *Distributed architectures*. In these architectures the system designer first analyzes the learning agent task and then splits it up into simpler tasks. Each of these tasks is then implemented as a monolithic architecture. Usually the simpler tasks identified by the system designer have an intuitive correspondence with the notion of *atomic behavioral module*, that is of a behavioral module which cannot be reasonably further decomposed. After atomic behavioral units are identified they must be interconnected to build the complete learning system; this can result in two different kind of architectural organizations, which are listed in the following.
  - *Flat architectures*. In flat architectures all the behavioral modules are basic behaviors, that is they are directly interfaced with the environment. In case two or more behavioral modules produce homogeneous responses (e.g., they control the same movement actuators) their

actions are composed by an appropriate composition module (see Figure 2a); otherwise they just send their responses to the appropriate actuators (see Figure 2b).

· *Hierarchical architectures*. In this case a hierarchy of behavioral modules is used to build the learning system. Beside basic behaviors, which directly connect the system to the external world, there are coordination behaviors, which are in charge of coordinating basic and other coordination modules. Figure 3 shows an example of a possible architecture of an agent implemented using ALECSYS. In this case we have three basic behaviors and two coordination behaviors. C1 is in charge of coordinating B1 and B2, while C2's task is to coordinate C1 and B3. For example C2 could decide that whenever C1 and B3 are proposing some action, C1 should have priority. In turn, C1 could learn to compose the actions proposed by B1 and B2 into an intermediate action. Say B1 proposed a movement in direction –10˚ and B2 a movement in direction +30˚; then C1 could have learnt to mediate the two proposals in a +20˚ movement (other solutions are possible in which the two proposed actions are given different weights).

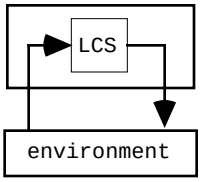


Figure 1. *Monolithic architecture.*

Figure 2. *Flat architectures.*

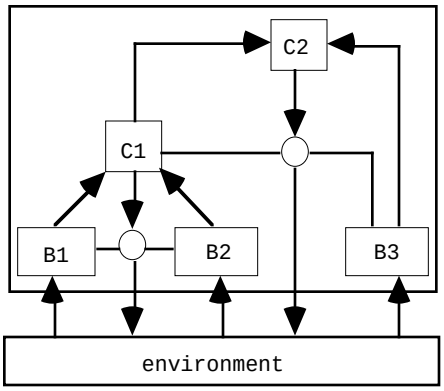


Figure 3. *An example of hierarchical architecture obtainable with ALECSYS.*

In ALECSYS, every single module is an enhanced version (see Dorigo, 1993) of a learning classifier system (CS) as proposed for example by Booker, Goldberg & Holland (1989). CSs are a rather complex paradigm for reinforcement learning. Functionally, they can be split in three components. The first one, called the *performance system*, is a kind of parallel production system; its role is to map input sensations into output actions. In the current version of ALECSYS, the performance system is a reactive system, in that internal messages are not allowed and therefore the system cannot remember past actions.

The second and third components, the *credit apportionment* and the *rule discovery* components respectively, are the learning components of a CS. The task of the credit apportionment subsystem is to evaluate rules in the rule base so that useful rules are ranked higher than less useful ones. For each rule a variable, called *strength*, measures the usefulness of the rule as evaluated by the credit apportionment subsystem. Strength is changed by means of redistribution of reinforcements received by the CS as feedback for actions performed. The algorithm we used is an extended version of the *bucket brigade* (Holland, 1986) which has been presented in details elsewhere (Dorigo, 1993). Since reinforcements are received at each step from an external trainer, ALECSYS is a *supervised* reinforcement learning system.

The third components is the rule discovery subsystem, which in ALECSYS is implemented by means of a genetic algorithm (GA). Genetic algorithms are a kind of evolutionary algorithm first proposed by Holland (1975). They work applying so-called genetic operators to a population of individuals which code solutions to a given problem. In the context of machine learning, most of the time individuals are rules and genetic operators mutate and recombine rules to produce new, hopefully more useful, ones. New rules, which overwrite low strength rules in the population, are tested and retained in case they demonstrate their utility to the learning system performance.

The main strengths of GAs are that:

- They can be easily implemented on a parallel computer (e.g., see Spiessens & Manderick, 1991).
- They are very efficient in recombining rule components, favoring the reproduction, and therefore the survival, of those components which are more often contained in rules with a higher than average strength. It has been proven that the number of structures which are processed by the GA at every cycle is much greater than the number of individuals in the population (Booker, Goldberg & Holland, 1989; Bertoni & Dorigo, 1993).
- They seem to be only slightly sensitive to the precision with which the usefulness of individuals, i.e. their strength, is evaluated. This is important because strength, which is evaluated by the bucket brigade, is only a rough indicator of good performance.

In ALECSYS, the GA is called when the apportionment of credit system has reached a steady state, i.e. when the strengths of rules in the population tend to be stationary (this property is monitored at runtime). It works applying in sequence the crossover and the mutation operators (Goldberg, 1989) and returning the modified population (Figure 4). More details about the actual implementation can be found in (Dorigo, 1993).

```
function   GA(Pop);
  Pop <- Crossover(Pop);
  Pop <- Mutate(Pop);
return
```

Figure 4. *The genetic algorithm.*

4. Experimental settings

When planning an experiment, the environment, the agent, and the target behavior must be designed together. Such entities are introduced here separately for descriptive convenience only.

4.1 Environment

What is a good experimental setting to show that proper sequences can emerge? Clearly, one in which: (i) agent-environment interactions are sequential, and (ii) the sequential nature of the interactions is not due to states of the environment. Indeed, under these conditions we have the guarantee that the relevant states are those of the agent. Therefore, we have carried out our initial experiments on sequential behavior in forgetful environments, that is in environments that keep no track of the effects of the agent's move.

Our environment is basically an empty space containing two objects, that we respectively call A and B (Figure 5). The distance between A and B, which lie on a bidimensional plane in which the agent can move freely, is approximately 100 forward steps of the agent. In some of the experiments, both objects emit a signal when the agent enters a circular area of predefined radium around the object (shown by the dashed circles in the figure).

4.2 The agent's "body"

The agent is a simulation of a simple mobile robot, which is intended to play the role of an artificial organism, and is thus called the *Animat* (see Wilson, 1987). The Animat's sensors are two on/off eyes with limited visual field of 180° and an on/off microphone. The eyes are able to detect the presence of an object in their visual fields, and can discriminate between the two objects A and B. The visual fields of the two eyes overlap by 90°, so that the total angle covered by the two eyes is 270°, and is partitioned in three areas of 90° each (see Figure 5).

The Animat's actuators are two independent wheels that can either stay still**,** move one or two steps forward, or one step backward.

4.3 Target behavior

The target behavior is the following: the Animat should approach object A, then approach object B, then approach object A, and so on. (A more complex target behavior has been adopted in the experiment reported in Section 7.) This target sequence can be represented by the regular expression $\{\alpha\beta\}^*$, where we denote by α the behavioral phase in which the Animat should approach object A, and by β the behavioral phase in which the Animat should approach object B. We assume that the transition from one phase to the next should occur when the Animat's microphone senses a sound, which acts as a transition signal. This signal tells the Animat that it is time to switch to the next phase, but does not tell it which phase should be the next.

Concerning the production of the transition signal, there are basically two possibilities:

- *external-based transition*: the transition signal is produced by an external source (e.g., the trainer) independently of the current interaction between the agent and its environment;
- *result-based transition*: the transition signal is produced when a given situation occurs as a result of the agent's behavior; for example, a transition signal is generated when the Animat

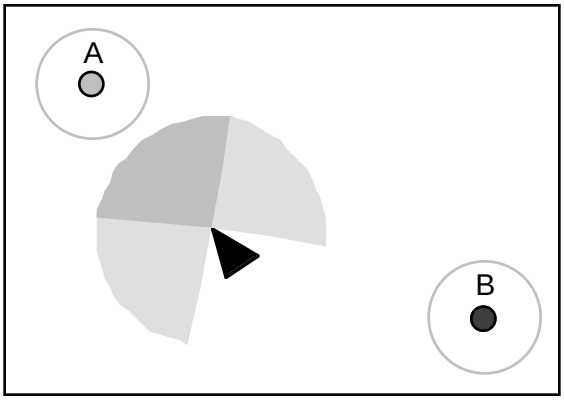


Figure 5. *The environment for sequential behavior.*

has come close enough to an object.

The choice between these two variants corresponds to two different intuitive conceptions of the overall task. If we choose external-based transitions, what we actually want from the Animat is that it learns to switch phase each time we tell it to. Instead, if we choose result-based transitions, we want the Animat to achieve a given result, and then to switch to the next phase. In fact, suppose that the transition signal is generated when the agent reaches a given threshold distance from A or from B. This means that we want the agent to reach object A, then to reach object B, and so on. As we shall see, the different conception of the task underlying this choice influences the way in which the Animat can be trained.

It would be easy to turn the environment described above into a Markov environment, so that a reactive agent could learn the target behavior. For example, we could assume that A and B are two lights, which are alternatively switched on and off, exactly one light being on at each moment. In this case, a reactive Animat could learn to approach the only visible light, and a pseudo-sequential behavior would emerge as an effect of the dynamic nature of the environment.

4.4 The agent's controller and sensorimotor interfaces

For the $\{\alpha\beta\}^*$ behavior we implemented two agents with different control architectures. We used a monolithic architecture, and a two-level hierarchical architecture (see Section 3). In this paper we report the experiments performed with the latter, which gave better results.

The two-level hierarchical architecture was organized as follows. Basic modules consisted of two independent CSs, that we shall call CS_α and CS_β, respectively in charge of learning the two basic behaviors α and β. The coordinator consisted of one CS, in charge of learning the sequential coordination of the lower level modules.

The input of each basic module represents the relative direction in which the relevant object is perceived. Given that the Animat's eyes partition the environment into four angular areas, both modules have a 2-bit sensory word as input. At any cycle, each basic module proposes a motor action, which is represented by 4 bits coding the movement of each independent wheel.

Coordination is achieved by choosing for execution exactly one of the actions proposed by the lower level modules. This choice is based on the value of a 1-bit word, that represents the internal state of the agent, and that we therefore call the *state word*. The effect of the state word is hardwired: when its value is 0, the action proposed by CS_α is executed; when the value is 1, it is CS_β that wins.

The coordinator receives as input the current value of the state word, and 1 bit representing the state of the microphone; this bit is set to 1 at the rising edge of the transition signal, and is equal to 0 otherwise. The possible actions for the coordinator are: (i) set the state word to 0, and (ii) set

the state word to 1. The task that the coordinator has to learn is to maintain the same phase if no transition signal is perceived, and to switch phase each time a transition signal is perceived.

The basic controller architecture is described in Figure 6. Viewed as a dynamic system, it is a Mealy machine (see Equations 1, Section 2), in that at each cycle t, the sensory input at t and the value of the state word at t jointly determine both the action performed at t and the value of the state word at t+1.

4.5 Experimental design

For each experiment reported in this paper we ran twelve independent trials, starting from random initial conditions. Each trial included:

- a *basic learning session* of 4,000 cycles, in which the two basic behaviors α and β were learned;
- a *coordinator learning session* of 12,000 cycles, in which learning of basic behaviors was switched off and only the coordinator was allowed to learn;
- a *test session* of 4,000 cycles, where all learning was switched off and the performance of the agent was evaluated.

In the learning sessions, the agent's performance $P_{learn}(t)$ at cycle t was computed for each trial as:

$$P_{learn}(t) = \frac{\text{Number of correct actions performed from cycle 1 to cycle } t}{t}$$

where an action is considered as correct if it is positively reinforced. The graph of $P_{learn}(t)$ for a single trial is called a *learning curve*. In the test session, the agent's performance P_{test} is

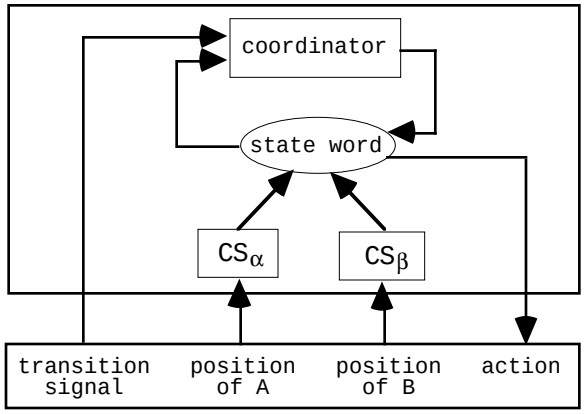


Figure 6. *Controller architecture of the Animat.*

measured for each trial as a single number:

$$P_{test} = \frac{\text{Number of correct actions performed in the test session}}{4000}$$

For each experiment we shall show the coordinator learning curve of a single, typical trial, and report the mean and standard deviation of the twelve P_{test} values for: (i) the two basic behaviors (α and β); and (ii) the two coordinator's tasks (*maintain* and *switch*). It is important to remark that the performance of the coordinator is judged from the overall behavior of the agent. That is, the only information available to evaluate such performance is whether the agent is actually approaching A or approaching B; no direct access to the coordinator's state word is allowed. Instead, to evaluate the performance of the basic behaviors it is also necessary to know at each cycle whether the action performed was suggested by CS_α or by CS_β; this fact is established by directly inspecting the internal state of the agent.

Finally, to compare the mean performances of a pair of experiments we use the two-tailed t-test, computing the probability p that the samples produced by the two experiments are drawn from populations with the same mean performance.

5. Training policies

We view supervised reinforcement learning as a mechanism to translate a specification of the agent's target behavior into a control program that realizes it (Dorigo & Colombetti, 1992). As the translation process takes place in the context of agent-environment interactions, the resulting control program is highly sensitive to features of the environment that would be very difficult to model explicitly in a handwritten program (Dorigo & Colombetti, 1993).

As usual in the field, reinforcements are provided to our learning agent by a computer program, that we call the *reinforcement program* (RP): it is the RP that embodies a specification of the target behavior. We believe that an important quality an RP should possess is to be highly *agent independent.* In other words, we want the RP to base its judgments on high-level features of the agent's behavior, without bothering too much about the details of such behavior. In particular, we want the RP to be as independent as possible from internal features of the agent, which are unobservable to an external observer. This requirement is reminiscent of the well-known methodological principle advocated by behaviorism, which states that only observable variables should be considered in behavior theory. In our case, there are no methodological preoccupation of this sort, also because in principle we *can* observe the internal states of artificial agents. However, the very same requirement seems to be a sensible engineering principle. In fact, an RP which is independent of events that are internal to the agent will be more abstract, general

and portable to different agents; in particular, it will be less sensitive to possible degradation of the agent's hardware.

Let us consider the $\{\alpha\beta\}^*$ behavior, where:

α = approach object A;

β = approach object B.

The transitions from α to β and from β to α should occur whenever a transition signal is perceived.

The first step is to train the Animat to perform the two basic behaviors α and β. This is a fairly easy task, given that the basic behaviors are instances of approaching responses, that can be produced by a simple reactive agent. The only difficulty is due to the fact that the Animat's world has hidden states: in fact, when an object is behind the Animat, it cannot be seen. The problem has been solved by training each CS to turn the Animat when it does not see the relevant object. This training technique and its results have been described elsewhere (see for example Dorigo & Colombetti, 1992).

After the basic behaviors have been learned, the next step is to train the Animat's coordinator to generate the target sequence. Before doing so, we have to decide how the transition signal is to be generated. We have experimented with both external-based and result-based transitions.

5.1 External-based transitions

Let us assume that coordinator training starts with phase α. The trainer rewards the Animat if it approaches object A, and punishes it otherwise. At random intervals, the trainer generates a transition signal. After the first transition signal is generated, the Animat is rewarded if it approaches object B, and punished otherwise; and so on.

Let us now suppose that in phase α, and without any transition signal, the Animat changes behavior. Clearly, as soon as the Animat starts approaching B, the trainer will give a punishment, because a change of phase occurred in absence of a transition signal. But then suppose that the Animat goes on approaching B. What should the trainer do? It would be incoherent to go on punishing the Animat, because it is now doing well: in fact, it is persisting with the same behavior in absence of a transition signal.

On the basis of these considerations, we have applied what we call a *flexible reinforcement program* (RP_{flex}), that is:

- start with phase α;
- in phase α, reward the Animat if it approaches A, and punish it otherwise; in phase β, reward the Animat if it approaches B, and punish it otherwise;

- change phase at each transition signal;
- if the Animat appears to change behavior in absence of a transition signal, punish it but change phase;
- analogously, if the Animat appears not to change behavior in presence of a transition signal, punish it but restore the previous phase.

The rationale of this reinforcement program is that the trainer punishes an inadequate treatment of the transition signal, but rewards coherency of behavior. Experiments 1, 2, and 3 (next section) have been run using RP_{flex}.

5.2 Result-based transitions

Let us now suppose that the target sequential behavior is understood as follows: the agent should approach and reach object A, then approach and reach object B, etc. A major difference with respect to the previous case is that a transition signal is now generated each time the agent comes close enough to an object (see the dashed circles in Figure 5). This calls for a different reinforcement program. In fact, it no longer makes sense for the trainer to flexibly change phase when the agent switches behavior: a phase is completed only when a given result is achieved, that is when the relevant object is reached.

We have therefore used a different reinforcement program, that we call the *rigid reinforcement program* (RP_{rig}):

- start with phase α;
- in phase α, reward the Animat if it approaches A, and punish it otherwise; in phase β, reward the Animat if it approaches B, and punish it otherwise;
- change phase at each transition signal, which is generated when the Animat gets to a predefined distance from the relevant object.

This program embodies the idea that the target behavior involves reaching objects, not just approaching them. However, the Animat *did not* learn the target behavior when trained with the RP_{rig}.

It is not difficult to understand why. Consider an interval $[t_1,t_2]$, in which no transition signal is produced, and assume that the Animat erroneously changes behavior at t_1. With the RP_{rig}, the Animat will be punished until it restores the previous behavior. But this means that in the interval $[t_1,t_2]$ the Animat will be punished if it keeps the same behavior, and rewarded if it changes behavior, even if no transition signal is perceived. From the Animat's point of view, this program

is incoherent: maintaining the same behavior in absence of a transition signal is sometimes rewarded, sometimes punished. In fact, the RP_{rig} rewards the Animat in three different cases[2]:

(i) when the Animat changes behavior in presence of a transition signal;

(ii) when the Animat *does not* change behavior in absence of a transition signal (provided its current behavior is the right one) ;

(iii) when the Animat *does* change behavior in absence of a transition signal (provided its current behavior is the wrong one).

Clearly, the problem is to make the agent distinguish between case (ii) and case (iii). To do so, it is sufficient to know at cycle N+1 whether the action performed at cycle N was right or wrong; and therefore, it is sufficient for the Animat to store the sign of the reinforcement received from the RP_{rig} at the previous cycle.

To allow the Animat to remember whether it had been rewarded or punished at the previous cycle, we introduced a 1-bit *reinforcement sensor*, that is a 1-bit field in the sensory interface telling the Animat whether the previous action had been rewarded or punished. In this way, the agent is able to develop specific behavior rules for case (iii), different from the rules for case (ii). Experiments 4, 5, and 6 (next section) show that the Animat is able to learn the target behavior when trained with the RP_{rig}, if its sensory interface includes the reinforcement sensor. We think that the notion of reinforcement sensor is not trivial, and therefore it needs to be discussed in some detail.

5.3 Meaning and use of the reinforcement sensor

At each moment, the reinforcement sensor stores information about what happened in the previous cycle, and as such contributes to the agent's dynamic behavior. Its characteristic feature is that it stores information about the behavior of the trainer, not of the physical environment. It may seem that such information is available to the Animat even without the reinforcement sensor, as it is received and processed by the credit apportionment module of ALECSYS. The point is that information about reinforcement is not available to the Animat's controller, unless it is coded into the sensory interface. To speak metaphorically, an agent endowed with the reinforcement sensor not only *receives* reinforcements, but also *perceives* them.

It is interesting to see how the information stored by the reinforcement sensor is exploited by the learning process. Let the reinforcement sensor be set to 1 if the previous action was rewarded, and to 0 if it was punished. When trained with the RP_{rig}, the Animat will develop behavior rules that can manage situation (iii) above, that is rules that change phase in absence of a transition

---

2 Analogous cases hold for punishments.

signal if the reinforcement sensor is set to 0. Such rules can be viewed as *error recovery rules*, in that they tell the agent what to do in order to fix a previous error in phase sequencing. Rules matching messages with the reinforcement sensor set to 1 will be called *normal rules*, to distinguish them from error recovery rules.

Without a reinforcement sensor, punishments are exploited by the system only to decrease the strength of a rule that leads to an error (i.e., to an incorrect action). With the reinforcement sensor, punishment are used for one extra purpose, that is to enable error recovery rule at the next cycle. In general, as learning proceeds less and less errors are made by the Animat, and the error recovery rules become increasingly weaker, so that sooner or later they are removed by the genetic algorithm.

Error recovery rules presuppose a reinforcement, and thus can be used only as far as the trainer communicates with the agent. If the trainer is switched off to test the acquired behavior, the reinforcement sensor must be clamped to 1, so that normal rules can be activated. This means that after we switch the trainer off, error recovery rules will remain silent; it is therefore advisable to do so only after all recovery rules have been eliminated by the genetic algorithm.

In the experiments reported in this paper, error recovery rules were either eliminated before we switched off the trainer, or they became so weak that they were practically no longer activated. With more complex tasks, however, one can easily imagine that some error recovery rules could maintain a strength high enough to survive and to contribute to the final behavior; in similar situations, switching off the trainer would actually impoverish the final performance. However, one could switch off the learning algorithm: the use of error recovery rules presupposes that an external system gives positive or negative "judgments" about the Animat's actions, but does not require the learning algorithm to be active. After switching off learning, the trainer actually turns into an *advisor*, that is an external observer in charge of telling the agent, which is no longer learning anything, whether it is doing well or not.

We still do not know whether the use of an advisor has interesting practical applications; it seems to us that it could be useful in situations where the environment is so unpredictable, that even the application of the most reasonable control strategy will frequently lead to errors. In a similar case, it would not be possible to avoid errors through further learning; error recovery seems therefore to be an appealing alternative.

6. Experimental results

In this section we report the results of the experiments on the $\{\alpha\beta\}^*$ behavior. The experiments described are the following:

- Experiments 1–3: sequential behavior with external-based transitions and flexible reinforcement program.
- Experiments 4–6: sequential behavior with result-based transitions, rigid reinforcement program and reinforcement sensor.

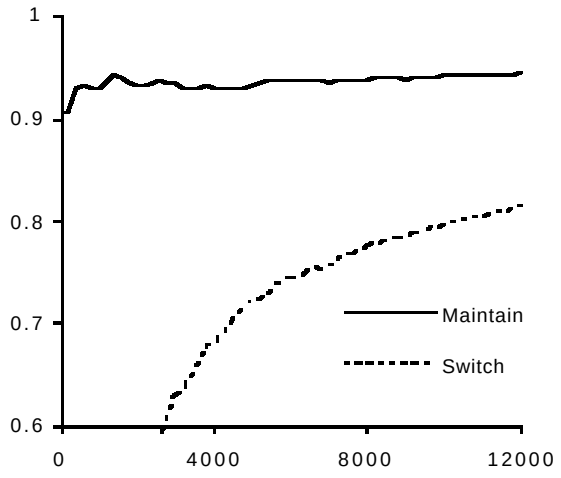
Experiments 1–3 and 4–6 are compared using two-tailed t-tests.

Experiment 1: External-based transitions and flexible reinforcement program

In this experiment, transition signals were produced randomly, with an average of one signal every 50 cycles. The Animat was trained with the flexible reinforcement program, RP_{flex}. Figure 7 shows a typical learning curve for the coordinator learning session, and reports the mean and standard deviation of the performances obtained in the test session out of twelve trials.

It appears that the Animat learns to maintain the current phase (in absence of a transition signal) better than to switch phase (when it perceives a transition signal). This result is easy to interpret: as transition signals are relatively rare, the Animat learns to maintain the current phase faster than it learns to switch phase.

As a whole, however, the performance of the coordinator is not fully satisfactory. One factor that keeps the performance of the coordinator well below 1 is that the performances of the two basic behaviors are not close enough to 1. In fact, during the training of the coordinator an action may be punished even if the coordinator has acted correctly, if a wrong move is proposed by the relevant basic CS.



	Task α	Task β
Mean	0.9155	0.8850
Std. deviation	0.0656	0.1192

Performance in test session: basic behaviors

	Maintain	Switch
Mean	0.9337	0.8276
Std. deviation	0.0471	0.0872

Performance in test session: coordination

Figure 7. *Experiment 1: Learning sequential behavior with external-based transitions.*

There is however another reason why the learning of the coordination tasks is not satisfactory: RP_{flex} cannot teach perfect coordination because there are ambiguous situations, that is situations where it is not clear whether the reinforcement program should reward or punish the agent. In fact, suppose that the Animat perceives a transition signal at cycle *N* when it is approaching A on a curvilinear trajectory like the one shown in Figure 8, and that at cycle *N*+1 it goes on following the same trajectory. By observing this behavior, RP_{flex} cannot know whether the Animat decided to go on approaching A, or whether it changed phase and is now turning to approach object B. As the agent's behavior is ambiguous, any reinforcement actually runs the risk of saying exactly the opposite of what it is intended to.

Ambiguous situations of the type described earlier arise because the agent's internal state is a hidden state from the point of view of the trainer. One possible solution is to make the relevant part of this state known to RP_{flex}. This was implemented in the next experiment.

Experiment 2: External-based transitions, flexible reinforcement program and agent-trainer communication

To eliminate ambiguous situations, we have simulated a communication process from the agent to the trainer: better reinforcements can be generated if the agent communicates its state to the reinforcement program, because situations like the one described earlier are no longer ambiguous.

To achieve this result, we added to the Animat the ability to assume two different observable states, that we conventionally call *colors*. The Animat can be either *white* or *black*, and can assume either color as the result of an action. In turn, the trainer can observe the Animat's color at any time. The basic modules are now able to perform one more action, that is to set a color bit to 0 (white) or to 1 (black). In the basic learning session, the Animat is trained not only to

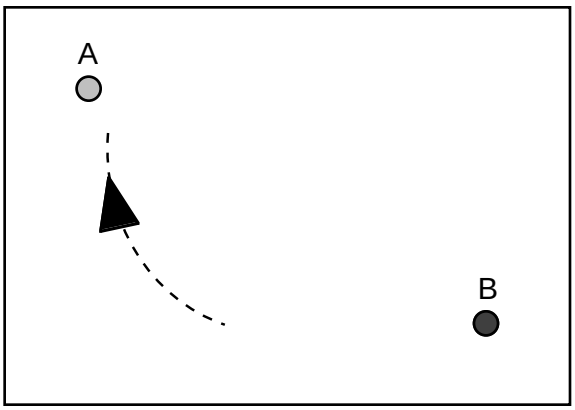


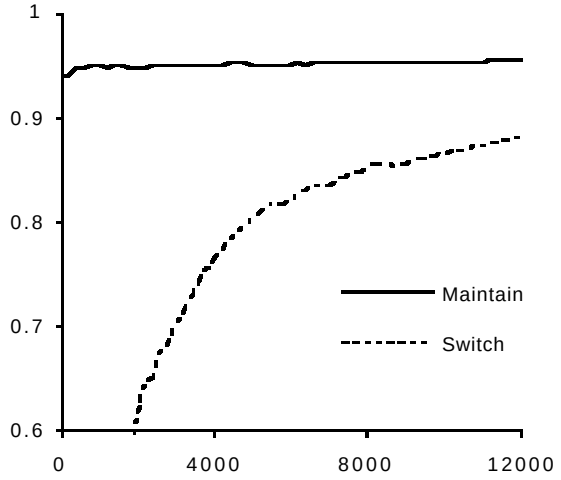
Figure 8. *An ambiguous situation.*

perform the approaching behaviors α and β, but also to associate a single color to each of them. During the coordinator learning session, RP_{flex} exploits information about the color to disambiguate the Animat's internal state, using the agent's color as a message.

As it emerges from the results of this experiment, reported in Figure 9, the coordinator's performance is slightly higher than in Experiment 1. However, this difference is only weakly significant as regards the switch task (two-tailed t-test: p = 0.094), and it is not significant as regards the maintain task (p = 0.340).

Experiment 3:External-based transitions, flexible reinforcement program and transfer of behavior

Another interesting solution to the problem of ambiguous situations is based on the notion of *behavior transfer*. The idea is that the {αβ}* behavior is based on two components: the ability to perform the basic behaviors α and β, and the ability to coordinate them in order to achieve the required sequence. While the basic behaviors are strongly linked to the environment, the coordination task is abstract enough to be learned in an environment, and then transferred to another one. Therefore, we proceeded as follows:



	Task α	Task β
Mean	0.8893	0.9442
Std. deviation	0.0650	0.0509

Performance in test session: basic behaviors

	Maintain	Switch
Mean	0.9493	0.8876
Std. deviation	0.0290	0.0809

Performance in test session: coordination

Figure 9. *Experiment 2: External-based transitions and agent-trainer communication.*

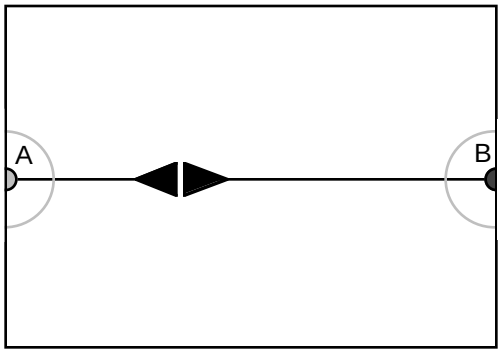
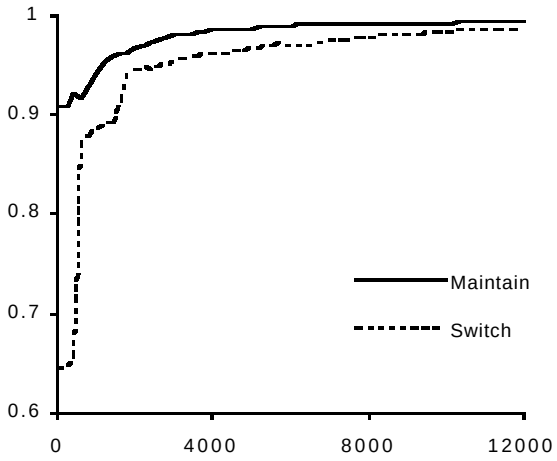


Figure 10. *The 1D environment.*

- The Animat learned the complete $\{\alpha\beta\}^*$ behavior in a simpler environment, where the ambiguity problem did not arise.
- Then, the Animat was then trained to perform the two basic behaviors in the target environment.
- Finally, the coordinator rules learned in the simpler environment were copied into the coordinator for the target task. To this purpose, we selected the rules that had the highest performance in the simpler environment; therefore, all twelve experiments in the target environment were run with the same coordinator.

The simpler, non ambiguous environment used for coordination training is sketched in Figure 10. It is a 1D counterpart of the target environment: the Animat can only move to the left or to the right on a fixed rail. At each instant, the Animat is either approaching A or approaching B: no ambiguous situations arise.

As reported in Figure 11, the performance achieved in the 1D environment was almost perfect, due to the simplicity of the task. Figure 12 shows the results obtained by transferring the coordinator to an Animat that had previously learned the basic behaviors in the 2D environment.



	Task α	Task β
Mean	0.9999	0.9999
Std. deviation	0.0003	0.0002

Performance in test session: basic behaviors

	Maintain	Switch
Mean	0.9996	0.9963
Std. deviation	0.0012	0.0066

Performance in test session: coordination

Figure 11. *Experiment 3: External-based transitions in the 1D environment.*

	Task α	Task β
Mean	0.9646	0.9649
Std. deviation	0.0367	0.0315

Performance in test session: basic behaviors

	Maintain	Switch
Mean	0.9615	0.9501
Std. deviation	0.0286	0.0364

Performance in test session: coordination

Figure 12. *Experiment 3: Transferring the external-based coordinator from the 1D to the 2D environment.*

As it can be seen, there was an improvement in the coordinator's performance with respect to Experiment 1, weakly significant for the maintain task ($p = 0.095$) and highly significant for the switch task ($p = 0.0002$), and also a significant improvement with respect to Experiment 2 as regards the switch task ($p = 0.023$), but not as regards the maintain task ($p = 0.310$). As a whole, the transfer of the coordinator gave the best results.

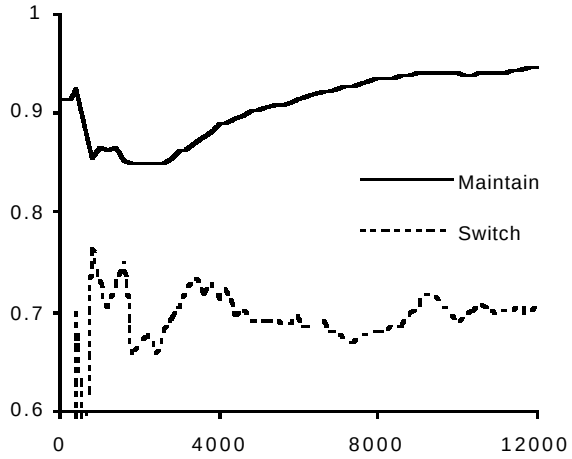
Experiment 4: Result-based transitions and rigid reinforcement program with reinforcement sensor

This experiment was run with result-based transitions: that is, the transition signal was generated each time the Animat reached an object. The target behavior was therefore conceived as: reach object A, then reach object B, and so on. Coherently with this view of the target behavior, we adopted the rigid reinforcement program, RP_{rig} (see Section 5.2); the Animat was therefore endowed with the 1-bit reinforcement sensor.

The results, reported in Figure 13, show that the target behavior was learnt; however, the performance of the switch task was rather poor.

Experiment 5:Result-based transitions, rigid reinforcement program and agent-trainer communication with reinforcement sensor

This experiment is the result-based analogous of Experiment 2: the Animat was trained to assume a color, thus revealing its internal state. The results are reported in Figure 14. Communication significantly improved the performance of both the maintain ($p = 0.016$) and the switch task ($p = 0.002$).



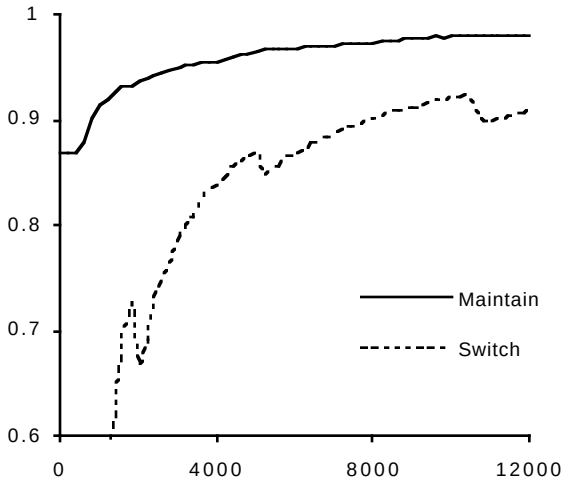
	Task α	Task β
Mean	0.9451	0.9743
Std. deviation	0.0426	0.0400

Performance in test session: basic behaviors

	Maintain	Switch
Mean	0.9313	0.7166
Std. deviation	0.0528	0.1559

Performance in test session: coordination

Figure 13. *Experiment 4: Learning result-based transitions.*



	Task α	Task β
Mean	0.9838	0.9879
Std. deviation	0.0201	0.0144

Performance in test session: basic behaviors

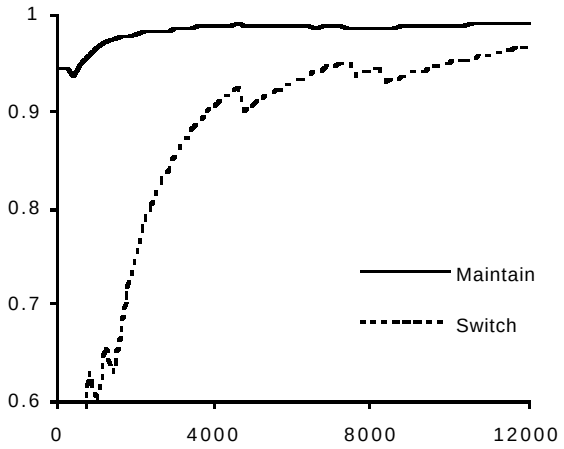
	Maintain	Switch
Mean	0.9789	0.8885
Std. deviation	0.0350	0.0788

Performance in test session: coordination

Figure 14. *Experiment 5: result-based transitions and agent-trainer communication.*

Experiment 6: Result-based transitions, rigid reinforcement program and transfer of behavior

This experiment is the result-based analogue of Experiment 3. Figure 15 shows the results obtained in the 1D environment, an d Figure 16 gives the performances of the Animat in the 2D environment, after transferring the best coordinator obtained in the 1D environment. The final performances were significantly better than in Experiment 4, both for the maintain ($p = 0.0009$) and for the switch task ($p = 0.0001$). In relation with Experiment 5, the improvement of the switch task was weakly significant ($p = 0.095$), while it was not significant for the maintain task ($p = 0.2818$).



	Task α	Task β
Mean	0.9999	0.9999
Std. deviation	0.0002	0.0001

Performance in test session: basic behaviors

	Maintain	Switch
Mean	0.9903	0.9670
Std. deviation	0.0317	0.0565

Performance in test session: coordination

Figure 15. *Experiment 6: Result-based transitions in the 1D environment.*

	Task α	Task β
Mean	0.9935	0.9895
Std. deviation	0.0067	0.0112

Performance in test session: basic behaviors

	Maintain	Switch
Mean	0.9903	0.9372
Std. deviation	0.0078	0.0559

Performance in test session: coordination

Figure 16. *Experiment 6: Transferring the result-based coordinator from the 1D to the 2D environment.*

We conclude this set of experiments with Figure 17, summarizing the results of the t-tests for all relevant pairs of experiments.

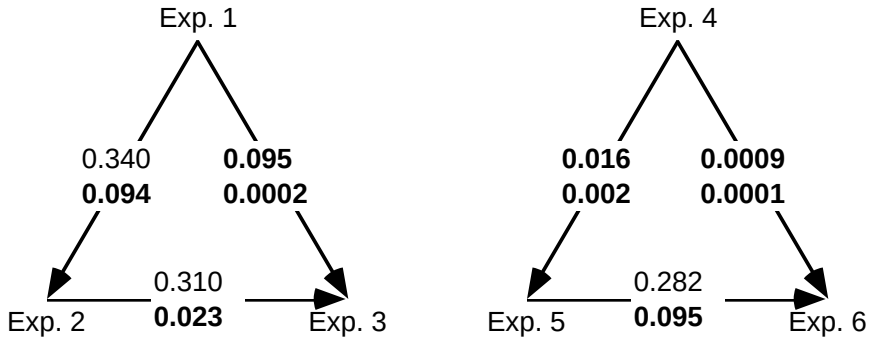


Figure 17. *Result of t-tests for the maintaining (above) and switching (below) tasks. Significant probability levels are in bold. Arrows indicate the direction of increasing value of the experimental mean; e.g. the mean of Exp.1 is lower of both the mean of Exp.2 and Exp.3.*

7. Conclusions

In this paper we have presented an approach to training an agent which learns proper behavioral sequences. Many other researchers have tackled the problem of learning sequences of actions in the realm of classifier systems (e.g., Riolo, 1989). Our work differentiate itself in that the building blocks of our sequences are elementary behaviors instead of simple actions.

We have discussed at length the difference between pseudo- and proper sequences, and we have shown that ALECSYS, our CS-based learning system, can learn proper sequences (pseudo-sequences were discussed in previous work, see Dorigo & Colombetti, 1992).

An important aspect of our research is the attention we pose on the interplay among the learner, the trainer, and the environment. We show that, when considering proper sequences, there are at least two kinds of transition signals which can cause a change to the next phase of the sequence: external-based transitions and result-based transitions. To each of these transition modalities corresponds a training policy, the flexible and the rigid training policies respectively. These policies require the introduction of communication features into our system:

- trainer-to-learner communication (through a reinforcement sensor, which makes explicitly available to the learner information about the quality of its behavior);
- learner-to-trainer communication (to let the learner know what is the current state of the learner).

Most interesting, the use of the reinforcement sensor introduces into the rule set a new kind of rules, called error recovery rules, which are activated only in case of punishment. These rules tend to disappear as learning goes on and performance improves.

Finally, we have shown that the coordination task, at least in the context of our experiments, is abstract enough that it can be learned in a simple situation, and then transferred into a more demanding one.

References

Beer, R. D., 1993. A dynamical systems perspective on autonomous agents. Submitted to *Artificial Intelligence*.

Bertoni, A., & M. Dorigo, 1993. Implicit Parallelism in Genetic Algorithms. *Artificial Intelligence*, 61, 2, 307–314.

Booker, L., D. E. Goldberg & J. H. Holland, 1989. Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, 40, 1-3, 235–282.

Colombetti, M., & M. Dorigo, 1992. Learning to control an autonomous robot by distributed genetic algorithms. *Proceedings of "From Animals To Animats," 2nd International Conference on Simulation of Adaptive Behavior (SAB92)*, Honolulu, Hawaii, MIT Press, 305–312.

Dorigo, M., 1992. Alecsys and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems. *Technical Report No.92-011*, Politecnico di Milano, Italy. (To appear in *Machine Learning*).

Dorigo, M., 1993. Genetic and non-genetic operators in ALECSYS. *Evolutionary Computation*, 1, 2, 149–162, MIT Press.

Dorigo M. & M. Colombetti, 1992. Robot Shaping: Developing Situated Agents through Learning. *Technical Report No. 92-040*, International Computer Science Institute, Berkeley, CA. (Submitted to *Artificial Intelligence*, August 1992).

Dorigo M. & M. Colombetti, 1993. Design and development of autonomous robots by reinforcement learning (in preparation).

Goldberg, D.E., 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley.

Holland, J. H., 1975. *Adaptation in natural and artificial systems*, The University of Michigan Press, Ann Arbor, Michigan.

Holland, J. H., 1986. Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-based Systems, in R.S.Michalski, J.G.Carbonell, & T.M.Mitchell (Eds.), *Machine Learning II*, Morgan Kaufmann.

Lin, L.-J., & T. M. Mitchell, 1992. Memory approaches to reinforcement learning in non-Markovian domains. *Technical Report CMU-CS-92-138,* School of Computer Science, Carnagie Mellon University, Pittsburgh, PA.

Littman, M.L., 1992. An optimization-based categorization of reinforcement learning environments. *Proceedings of "From Animals To Animats", 2nd International Conference on Simulation of Adaptive Behavior (SAB92)*, Honolulu, Hawaii, MIT Press, 262–270.

Mahadevan, S., & J. Connell, 1992. Automatic programming of behavior-based robots using reinforcement learning, *Artificial Intelligence*, 55, 2, 311–365.

McCluskey, E.J., 1986. *Logic Design Principles.* Prentice-Hall.

Riolo R.L., 1989. The emergence of coupled sequences of classifiers. *Proceedings of the Third International Conference on Genetic Algorithms*, J.D. Schaffer (Ed.), Morgan Kaufmann, 256–264.

Rosenschein, S. J., & L. P. Kaelbling, 1986. The synthesis of digital machines with provable epistemic properties. In J. Halpern, ed., *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge*, Morgan Kaufmann, Los Altos, CA, 83–98.

Singh, S. P., 1992. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8, 3-4, 323–339.

Spiessens, P., & B. Manderick, 1991. A massively parallel genetic algorithm: Implementation and first analysis. *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morkan Kaufmann, 279–286.

Watkins, C.J.C.H., 1989. *Learning with delayed rewards*. Ph. D. dissertation, Psychology Department, University of Cambridge, England.

Watkins, C.J.C.H., & P. Dayan, 1992. Technical Note: Q-learning. *Machine Learning*, 8, 3-4, 279–292.

Whitehead, S. D., & L. J. Lin, 1993. Reinforcement learning in non-Markov environments. Submitted to *Artificial Intelligence*.

Wilson, S., 1987. Classifier systems and the Animat problem. *Machine Learning*, 2, 3, 199–228.

Wilson, S., 1990. The Animat path to AI. *Proceedings of "From Animals To Animats," 1st International Conference on the Simulation of Adaptive Behavior (SAB90),* Cambridge, MA, MIT Press, 15–21.