Training Agents to Perform Sequential Behavior

Marco Colombetti and Marco Dorigo

Adaptive Behavior 1994; 2; 247
DOI: 10.1177/105971239400200302

Training Agents to Perform Sequential Behavior

Marco Colombetti*
Politecnico di Milano

Marco Dorigo†‡
Université Libre de Bruxelles

This article is concerned with training an agent to perform sequential behavior. In previous work, we have been applying reinforcement learning techniques to control a reactive agent. Obviously, a purely reactive system is limited in the kind of interactions it can learn. In particular, it can learn what we call pseudosequences*—that is, sequences of actions in which each action is selected on the basis of current sensory stimuli. It cannot learn proper sequences, in which actions must be selected also on the basis of some internal state. Moreover, it is a result of our research that effective learning of proper sequences is improved by letting the agent and the trainer communicate. First, we consider trainer-to-agent communication, introducing the concept of* reinforcement sensor*, which lets the learning robot explicitly know whether the last reinforcement was a reward or a punishment. We also show how the use of this sensor makes* error recovery *rules emerge. Then we introduce agent-to-trainer communication, which is used to disambiguate ambiguous training situations—that is, situations in which the observation of the agent's behavior does not provide the trainer with enough information to decide whether the agent's move is right or wrong. We also show an alternative solution to the problem of ambiguous situations, which involves learning to coordinate behavior in a simpler, unambiguous setting and then transferring what has been learned to a more complex situation. All the design choices we make are discussed and compared by means of experiments in a simulated world.*

Key Words: classifier systems; genetic algorithms; training; sequential behavior; autonomous agents

Introduction

In this article, we explore 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

* Progetto di Intelligenza Artificiale e Robotica, Dipartimento di Elettronica e Informazione, Piazza Leonardo da Vinci, 32, 20133 Milano, Italy; E-mail: colombet@elet.polimi.it

† IRIDIA, Avenue Franklin Roosevelt 50, CP 194/6, 1050 Bruxelles, Belgium; E-mail: mdorigo@ulb.ac.be

‡ On leave from Progetto di Intelligenza Artificiale e Robotica, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy; dorigo@elet.polimi.it.

in partially unpredictable environments. Most often, agents are viewed as *reactive systems*—that is, as systems whose actions are completely determined by current sensory 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 and Connell, 1992; Dorigo and Colombetti, 1994); 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 nonreactive 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 article, we present a different approach to the problem of learning sequential behavior patterns, viewed as the result of coordinating separately learned basic behaviors (Colombetti and 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 module being a learning classifier system (Booker, Goldberg, and Holland, 1989). The system runs in parallel on a network of transputers and has been connected to both simulated agents and physical robots. Although, in principle, ALECSYS allows one to train agents through delayed reinforcements, in the work presented here the behavior of agents has been shaped through a step-by-step reinforcement scheme, consisting of reinforcements provided after each action by an external trainer observing the agent's behavior.

Our general methodology is as follows: First, we define an environment, an agent, and a target behavior 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 wherein 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 determine whether the target behavior emerges and to analyze the effect of different design choices on the agent's performance.

In this article, we present the results of a research effort 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.

Reactive and Dynamical Behavior

As we have already pointed out, the simplest class of agents is that of *reactive systems*, agents that 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 sensory input $s(t)$ at time t:

$$a(t) = f[s(t)]$$

As argued by Whitehead and Lin (1994), reactive systems are perfectly adequate to Markov environments or, more specifically, when (1) the *greedy control strategy* is globally optimal, which means that choosing the locally optimal action in each environmental situation leads to a course of actions that is globally optimal, and (2) 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, such as Q-learning (Watkins, 1989; Watkins and Dayan, 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 Markovian environments, very often an agent cannot be assumed to have complete knowledge about the effects or costs of its own actions. Non-Markov situations are basically of two different types, hidden-state environments and sequential behavior.

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

Regarding *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 has to remove the object. If the agent is able to perceive that the object is in the given position, it will be able to sequence appropriately the placing and removing actions. However, a reactive agent will be unable to act properly at time t if (1) 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 (2) 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 *dynamical* if the action $a(t)$ it performs at time t depends not only on its current sensory input $s(t)$ but also on its state $x(t)$ at time t; in turn, such state and the current sensory input determine the state at time $t + 1$:[1]

$$\begin{aligned} a(t) &= f(s(t), x(t)) \\ x(t+1) &= g(s(t), x(t)) \end{aligned} \quad \textbf{(1)}$$

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

An 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 (or because) of their rather intuitive meaning, terms such as *memory* or *representation* can easily be used in a confusing way. Take, for example, the packing task proposed by Lin and Mitchell (1992) as an example of a non-Markovian 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. (p. 1)

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 "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.

Let's 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 trace 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 this environment is *forgetful*, in that it does not keep track of

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

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 such states do not exist. However, we still have two possible interpretations:

- Internal states are memories of past actions. State 0 means that B has just been touched, and state 1 means that A has just been touched.
- Internal states are goals, determining the agent's current task. State 0 means that the current task is to reach A, whereas state 1 means that the current task is to reach B.

The conclusion we draw is that terms such as *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 article, 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 dynamical agents possessing internal states by design, and which learn to use them to produce sequential behavior. Then, if we interpret internal states as goals, this amounts to learning an action plan, able to enforce the correct sequencing of actions. However, this intuitive idea must be considered with care.

Not all behavior patterns that initially appear to be based on an action plan are necessarily dynamical. Consider 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, and so on. This sequential 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	and prey ahead	→ move ahead.
Rule 3:	Not grasped	and prey at contact	→ grasp.
Rule 4:	Grasped	and nest ahead	→ move ahead.
Rule 5:	Grasped	and in nest	→ drop.

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

It is interesting to see why the behavior pattern just described, 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 dynamical system constituted by the agent and the environment, the task is actually dynamical. The relevant states are the states of the environment, which keep 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 text, we shall call *pseudosequences* those tasks performed by a reactive agent that are sequential by virtue of the dynamical nature of the environment, and we shall reserve the term *proper sequence* for tasks that can be executed only by dynamical agents, by virtue of their internal states.

Let us rephrase the preceding considerations. As has already been suggested in the literature (see, for example, Rosenschein and Kaelbling, 1986; Beer, 1994), 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 dynamical, 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 are 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 article, rather than experimenting with sequences of single actions, we have focused on tasks made up of a sequence of phases, where a *phase* is a subtask that 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 decision to be made 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, which we call *basic behavioral modules*, are directly connected with

sensory and motor 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 module, called *behavior coordination modules*, whose main task is to learn to coordinate other behaviors. Coordination modules are connected to lower-level modules, both basic and coordination ones, and either can choose which of the actions proposed by connected modules should be given priority or can compose such actions into a complex behavioral response (Dorigo and Schnepf, 1993).

There are different ways in which behavioral modules can be put together to build a learning system. In a recent study (Dorigo and Colombetti, 1994) we investigated a number of them, which are briefly summarized here:

1. *Monolithic architecture.* In this architecture (Fig. 1), there is only one learning module, 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 that first motivated distributed architectures.
2. *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 an *atomic behavioral module*—that is, of a behavioral module that cannot reasonably be further decomposed. After atomic behavioral units are identified, they must be interconnected to build the complete learning system; this can result in two different kinds of architectural organizations:

    a. *Flat architectures.* In flat architectures, all the behavioral modules are basic—that is, they are directly interfaced with the environment. In the event that two or more behavioral modules produce homogeneous responses (e.g., they control the same actuators), their actions are composed by an appropriate composition module (Fig. 2a); otherwise, they just send their responses to the appropriate actuators (Fig. 2b).

    b. *Hierarchical architectures.* In hierarchical architectures, a hierarchy of modules is used to build the learning system. Aside from basic behavioral modules, which directly connect the system to the external world, there are behavior coordination modules, 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 behavioral modules and two coordination modules: C1 is in charge of coordinating B1 and B2, whereas C2's task is to coordinate C1 and B3. For example, C2 could decide that whenever C1 and B3 propose an action, C1 should have priority.

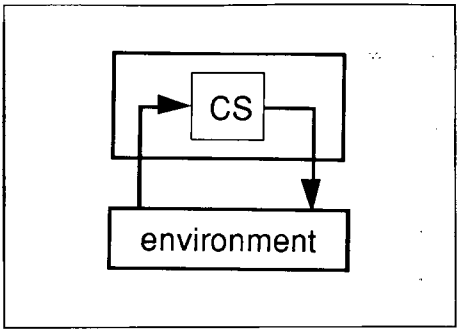


Figure 1
Monolithic architecture. CS, learning classifier system.

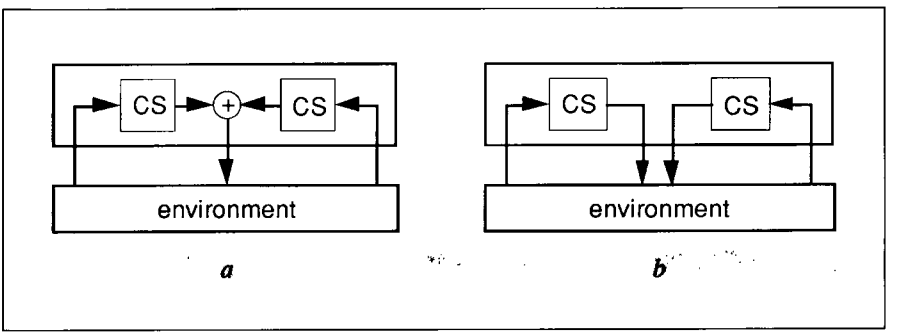


Figure 2
Flat architectures. CS, learning classifier system.

> 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 degrees and B2 a movement in direction +30 degrees; then C1 could learn to mediate the two proposals into a +20 degree movement (other solutions are possible in which the two proposed actions are given different weights).

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, and 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 version of ALECSYS used for the present work, 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, respectively called *credit apportionment* and *rule discovery*, are the learning components of a CS. The task of the credit apportionment

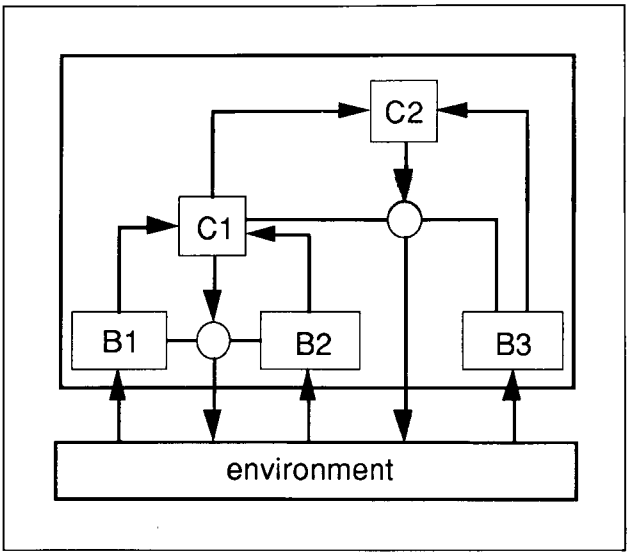


Figure 3
An example of hierarchical architecture obtainable with ALECSYS.

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 detail elsewhere (Dorigo, 1993). In this article, we use ALECSYS as a step-by-step reinforcement learning system, as reinforcements are received at each step from an external trainer.

The third component is the rule discovery subsystem, which in ALECSYS is implemented by means of a genetic algorithm (GA). GAs are a kind of evolutionary algorithm first proposed by Holland (1975). They work by applying so-called genetic operators to a population of individuals that 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's performance.

The main strengths of GAs, within the framework of a CS, are that:

- They can be easily implemented on a parallel computer (e.g., see Spiessens and Manderick, 1991; Dorigo and Sirtori, 1991).
- They are very efficient in recombining rule components, favoring the reproduction, and therefore the survival, of those components that are

more often contained in rules with a higher-than-average strength. It has been proved that the number of structures that are processed by the GA at every cycle is much greater than the number of individuals in the population (Booker, Goldberg, and Holland, 1989; Bertoni and Dorigo, 1993).

- They seem to be only slightly sensitive (Fitzpatrick and Grefenstette, 1988) to the precision with which the usefulness of rules—that is, 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 the credit system has reached a steady state, that is, when the strengths of rules in the population tend to be stationary (this property is monitored at run time). It works applying in sequence the crossover and the mutation operators (Goldberg, 1989) and returning a modified population of rules. More details about the actual implementation can be found in Dorigo (1993).

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

A good experimental setting in which to show that proper sequences can emerge clearly is one in which (1) agent-environment interactions are sequential and (2) 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, ones that keep no track of the effects of the agent's move.

Our environment is basically an empty space containing two objects, which we respectively call *A* and *B* (Fig. 4). 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 radius around the object (shown by the dashed circles in the figure).

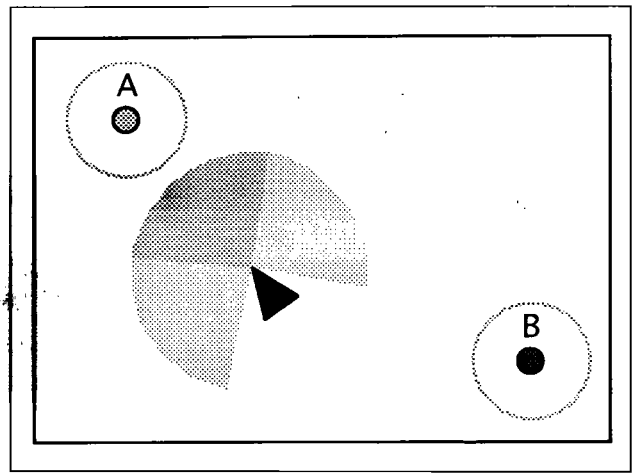


Figure 4
The environment for sequential behavior.

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 a limited visual field of 180 degrees 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 degrees, so that the total angle covered by the two eyes is 270 degrees and is partitioned into three areas of 90 degrees each (see Fig. 4). The animat's actuators are two independent wheels that can stay still, move one or two steps forward, or move one step backward.

4.3 Target behavior

The target behavior is as follows: The animat should approach object A, then approach object B, then approach object A, and so on. 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 senses 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.

There are basically two possibilities for the production of the transition signal. In *externally 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. In *result-based transition*, the transition signal is produced when a given

situation occurs as a result of the agent's behavior (e.g., a transition signal is generated when the animat 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 externally based transitions, what we actually want from the animat is that it learn to switch phase each time we tell it to do so. If instead 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 conceptions of the task underlying this choice influence the way in which the animat can be trained.

It would be easy to turn the environment just described into a Markovian 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 alternately 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 pseudosequential behavior would emerge as an effect of the dynamical 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 article, 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 two-bit sensory word as input. At any cycle, each basic module proposes a motor action, which is represented by four bits coding the movement of each independent wheel (two bits code the four possible movements of the left wheel, and two bits those of the right 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 one-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 its value is 1, it is CS_β that wins.

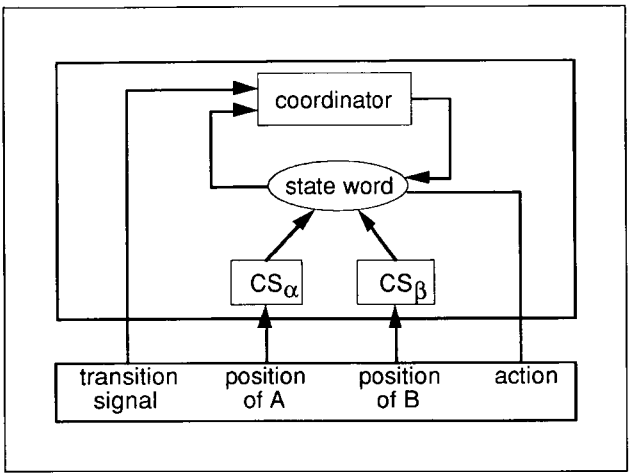


Figure 5
Controller architecture of the animat. CS, learning classifier system.

The coordinator receives as input the current value of the state word and one bit representing the state of the transition signal sensor; 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 to set the state word to 0 and to set the state word to 1. The tasks that the coordinator has to learn are to maintain the same phase if no transition signal is perceived and to switch phase each time a transition signal is perceived.

The controller architecture is described in Figure 5. Viewed as a dynamical system, it is a Mealy machine (see Equation 1, section 2), in that at each cycle t, the sensory input 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 article, we ran twelve independent trials, starting from random initial conditions. Each trial included a *basic learning session* of 4000 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; and a *test session* of 4000 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 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 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 both the two basic behaviors (α and β) and 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 also is 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 establish whether different experiments result in significantly different performances, we compute the probability (p) that the sample of performances produced by the different experiments are drawn from the same population. Following a widely adopted convention, a difference in performance is considered highly significant when $p < .01$; significant when $.01 \leq p < .05$; and weakly significant when $.05 \leq p < .1$. To compute p, we use the Kruskal-Wallis test to compare groups of more than two experiments and the Mann-Whitney test to compare pairs of experiments. These tests are the nonparametric counterparts of the more popular ANOVA (analysis of variance) and Student's t test for independent samples, respectively. The choice of nonparametric statistics is motivated by the fact that our data do not meet the requirements for their more powerful parametric counterparts (e.g., normal distribution).

5 Training Policies

We view training by reinforcement learning as a mechanism to translate a specification of the agent's target behavior, embodied in the reinforcement policy, into a control program that realizes it (Dorigo and Colombetti, 1994). As the learning mechanism carries out the translation in the context of agent-environment interactions, the resulting control program can be highly sensitive to features of the environment that would be difficult to model explicitly in a handwritten control program.

As usual in the field, reinforcements are provided to our learning agent by a computer program, which we call the *reinforcement program* (RP). It is therefore the RP that embodies the specification of the target behavior. We believe it is important for an RP 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. The very same requirement seems to be a sensible engineering principle, although we *can* observe the internal states of our artificial agents. An RP that is independent of events internal to the agent will be more abstract, general, and portable to different agents; in addition, it will be less sensitive to possible degradation of the agent's hardware.

Let us consider the $\{\alpha\beta\}^*$ behavior, where α means "approach object A" and β means "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 lies in 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 and Colombetti, 1994).

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 externally based and result-based transitions.

5.1 Externally 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 is punished otherwise; and so on.

Let us now suppose that in phase α the animat changes behavior in absence of any transition signal. Clearly, as soon as the animat starts approaching B, the trainer will administer a punishment, because a change of phase is occurring without a transition signal being produced. However, suppose the animat goes on approaching B. What should the trainer do? It would be incoherent to continue punishing the animat, because it is now doing well: That is, it is persisting with the same behavior in the absence of a transition signal.

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

- 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 the absence of a transition signal, punish it but change the phase.
- Analogously, if the animat appears not to change behavior in the 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 (discussed in the next section) were 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, and so on. 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 4). This calls for a different reinforcement program. In fact, it no longer makes sense for the trainer to change phase flexibly 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, which 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 a time 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 maintains 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 the absence of a transition signal is sometimes rewarded, sometimes punished. In fact, the RP_{rig} rewards the animat in three different cases[2]:

1. When the animat changes behavior in the presence of a transition signal
2. When the animat does not change behavior in the absence of a transition signal, provided its current behavior is the right one
3. When the animat does change behavior in the absence of a transition signal, provided its current behavior is the wrong one

Clearly, the problem is to make the agent distinguish between cases 2 and 3. To do so, it is sufficient to know at cycle $t + 1$ whether the action performed at cycle t 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 one-bit *reinforcement sensor*—that is, a one-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 3, different from the rules for case 2. Experiments 4, 5, and 6 (discussed in the 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 believe that the notion of a 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, it contributes to the agent's dynamical 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 no information about reinforcement is 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.

2 Analogous cases hold for punishments.

It is interesting to note 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 case 3 (cited previously)—that is, rules that change phase in the absence of a transition 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, punishments are used for one extra purpose, to enable error recovery rules at the next cycle. In general, as learning proceeds, fewer and fewer errors are made by the animat, and the error recovery rules become increasingly weaker, so that sooner or later they are removed by the GA.

Error recovery rules presuppose a reinforcement and thus can be used only as far as the trainer is on. 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 GA.

In the experiments reported in this article, error recovery rules either were 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*, an external observer in charge of telling the agent, which is no longer learning anything, whether or not it is doing well.

We do not know yet 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; therefore, error recovery seems to be an appealing alternative.

Experimental Results

The results of the experiments on the $\{\alpha\beta\}^*$ behavior are reported for sequential behavior with externally based transitions and a flexible reinforcement program (experiments 1–3) and sequential behavior with result-based transitions, a rigid reinforcement program, and a reinforcement sensor (experiments 4–6). The performances of the coordinator tasks (*maintain* and *switch*) of experiments 1 through 3 and experiments 4 through 6 then are compared using the Kruskal-Wallis test. When this test signals a significant difference, the performances are then pairwise compared through Mann-Whitney tests.

Experiment 1: Externally based transitions and flexible reinforcement program

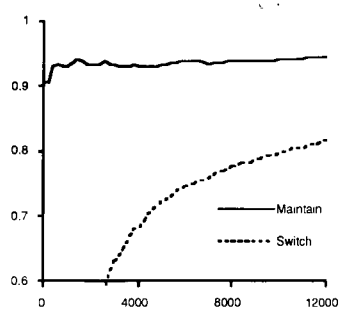
In experiment 1, transition signals were produced randomly, with an average of one signal every 50 cycles. The animat was trained with the flexible reinforcement program, RP_{flex}. Figure 6 shows a typical learning curve for the coordinator learning session and reports the mean and standard deviation of the performances obtained in the test session over 12 trials.

It appears that the animat learns to maintain the current phase (in the absence of a transition signal) better than it learns 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.

On the whole, however, the performance of the coordinator is not fully satisfactory, at least as far as the switch task is concerned. 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, assuming a wrong move is proposed by the relevant basic CS.

Another reason that the learning of the coordination tasks is not satisfactory is that RP_{flex} cannot teach perfect coordination because there are ambiguous situations, situations in which it is not clear whether the reinforcement program should reward or punish the agent. Suppose that the animat perceives a transition signal at cycle t when it is approaching A on a curvilinear trajectory such as the one shown in Figure 7 and that at cycle $t + 1$ 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 the opposite of what is intended.

Ambiguous situations of the type just described arise because the agent's internal state is hidden from the point of view of the trainer. One possible solution is to



	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 6
Experiment 1: Learning sequential behavior with externally based transitions.

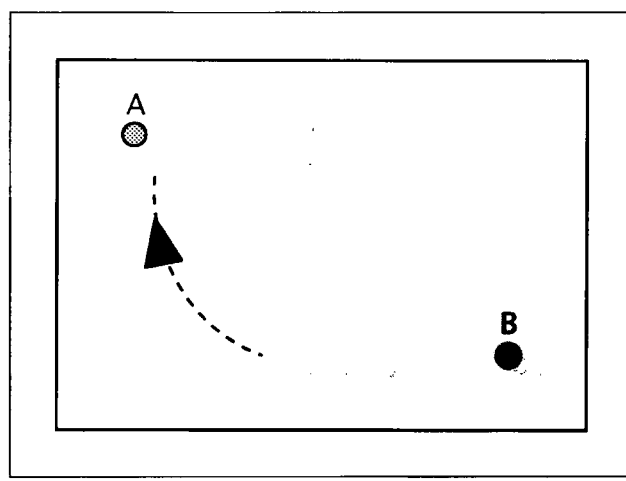
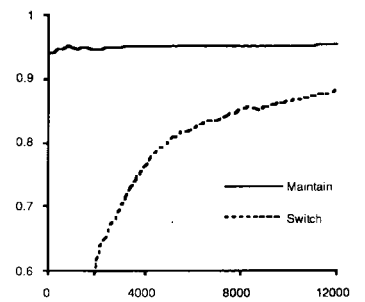


Figure 7
An ambiguous situation.

make the relevant part of this state known to RP_{flex}. This was implemented in the next experiment.

Experiment 2: Externally based transitions, a flexible reinforcement program, and agent-to-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 such as the one described earlier are no longer ambiguous.



	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 8
Experiment 2: Externally based transitions and agent-to-trainer communication.

To achieve this result, we added to the animat the ability to assume two different observable states, which 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 perform the approaching behaviors α and β but also to associate a single color to each of them. During the coordinator learning session, $\mathrm{RP}_{\mathrm{flex}}$ exploits information about the color to disambiguate the animat's internal state, using the agent's color as a message. The results of this experiment are reported in Figure 8.

Experiment 3: Externally based transitions, a flexible reinforcement program, and transfer of the coordinator

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

- The animat learned the complete $\{\alpha\beta\}^*$ behavior in a simpler environment, where the ambiguity problem did not arise.
- The animat was then trained to perform the two basic behaviors in the target environment.

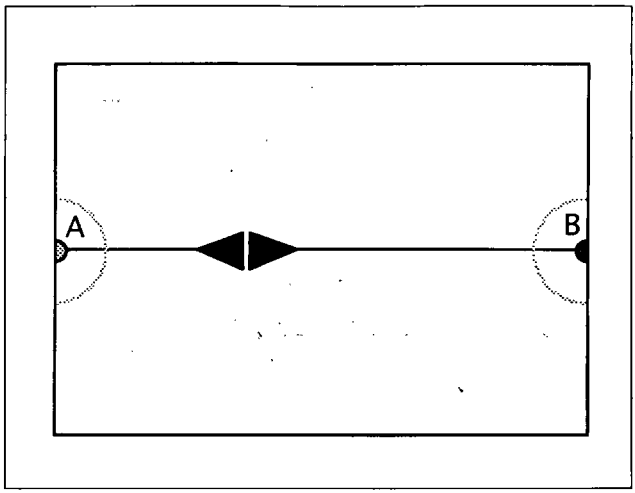


Figure 9
The one-dimensional environment.

- 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 12 experiments in the target environment were run with the same coordinator.

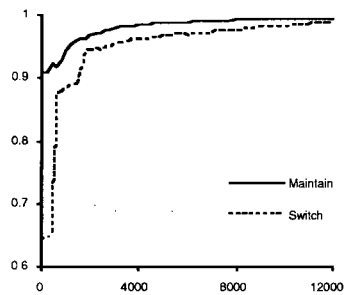
The simpler, nonambiguous environment used for coordination training is sketched in Figure 9. It is a one-dimensional counterpart of the target environment: The animat can move only to the left or to the right on a fixed rail. At each instant, the animat is either approaching A or approaching B; no ambiguous situations arise.

As reported in Figure 10, the performance achieved in the one-dimensional environment was almost perfect owing to the simplicity of the task. Figure 11 shows the results obtained by transferring the coordinator to an animat that had previously learned the basic behaviors in the two-dimensional environment.

Comparison of Experiments 1 through 3

For experiments 1 through 3, the Kruskal-Wallis test applied to the performances of the coordinator's tasks revealed no significant difference for the maintain task ($p = .2291$) but a highly significant difference for the switch task ($p = .0029$). We have therefore applied the Mann-Whitney test to the three pairs of performances of the switch task, obtaining the following results:

- A weakly significant difference between experiments 1 and 2 ($p = .0996$)



	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 10
Experiment 3: Externally based transitions in the one-dimensional 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

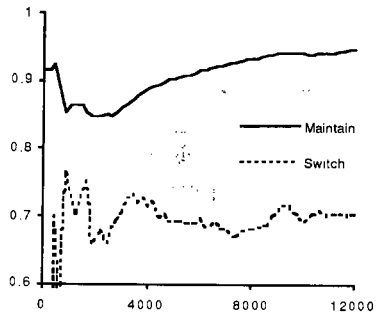
Figure 11
Experiment 3: Transferring the externally based coordinator from the one-dimensional to the two-dimensional environment.

- A highly significant difference between experiments 1 and 3 ($p = .0018$)
- A significant difference between experiments 2 and 3 ($p = .0243$)

Taking into account the mean performances of the switch task in the three experiments (see Figs. 7, 8, and 11), we conclude that with externally based transitions the maintain task is learned reasonably well, and with no significant difference, by all the strategies with which we have experimented. In contrast, for the switch task, the three strategies give different results. The best coordinator is obtained through transfer of the coordinator, although agent-to-trainer communication significantly overcomes the problem of learning in ambiguous situations.

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

Experiment 4 was run with result-based transitions: A transition signal was generated each time the animat reached an object. The target behavior was therefore conceived



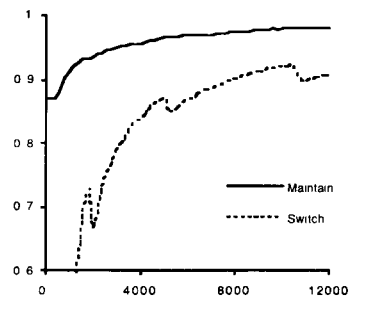
	Task α	Task β
Mean	0.9451	0.9743
Std. deviation	0.0426	0.0400

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 12
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

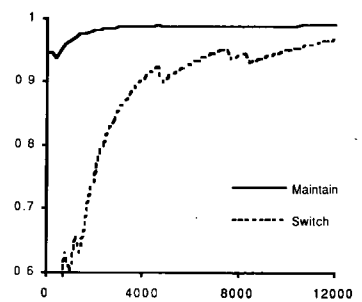
Performance in test session: coordination

Figure 13
Experiment 5: Result-based transitions and agent-to-trainer communication.

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 one-bit reinforcement sensor. The results, reported in Figure 12, show that the target behavior was learned, but the performance of the switch task was rather poor.

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

Experiment 5 is the result-based analog of experiment 2. The animat was trained to assume a color, thus revealing its internal state. The results are reported in Figure 13.



	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 14
Experiment 6: Result-based transitions in the one-dimensional 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 15
Experiment 6: Transferring the result-based coordinator from the one-dimensional to the two-dimensional environment.

Experiment 6: Result-based transitions, a rigid reinforcement program, and transfer of the coordinator

Experiment 6 is the result-based analog of experiment 3. Figure 14 displays the results obtained in the one-dimensional environment, and Figure 15 gives the performances of the animat in the two-dimensional environment, after transferring the best coordinator obtained in the one-dimensional environment.

Comparison of experiments 4 through 6

For experiments 4 through 6, the Kruskal-Wallis test applied to the performances of the coordinator's tasks demonstrated a highly significant difference for the maintain task ($p = .0002$) and a highly significant difference for the switch task ($p = .0002$). We have therefore applied the Mann-Whitney test to the three pairs of performances of both the maintain and the switch tasks, obtaining the following results:

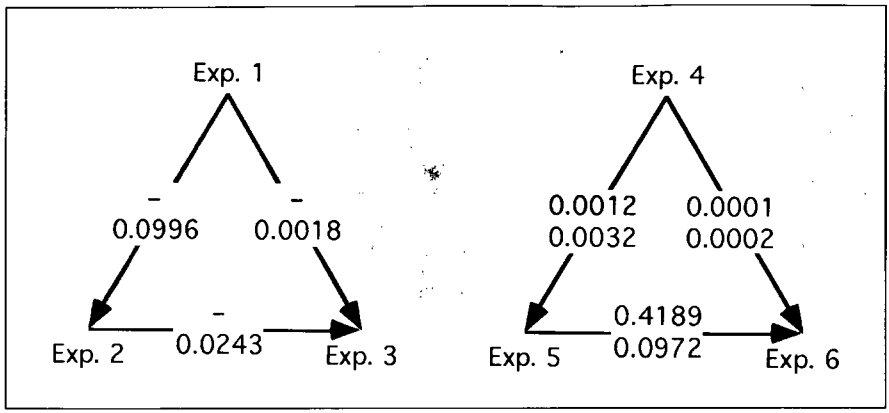


Figure 16
Results of Mann-Whitney tests for the maintain (above) and switch (below) tasks. Arrows indicate the direction of increasing value of the mean performance (e.g., the mean of experiment 1 is lower than the mean of experiments 2 and 3).

- For experiments 4 and 5, a highly significant difference for both the maintain task ($p = .0012$) and the switch task ($p = .0032$)
- For experiments 4 and 6, a highly significant difference for both the maintain task ($p = .0001$) and the switch task ($p = .0002$)
- For experiments 5 and 6, no significant difference for the maintain task ($p = .4189$) and a weakly significant difference for the switch task ($p = .0972$).

Taking into account the mean performances of the two tasks in these experiments (see Figs. 12, 13, and 15), we conclude that, with result-based transitions, coordinator transfer and agent-to-trainer communication are roughly equivalent. They significantly overcome the problem of learning in ambiguous situations for both the maintain and the switch tasks. Figure 16 summarizes the results of the Mann-Whitney tests for all relevant pairs of experiments.

7 Conclusions

In this article, we have presented an approach to training an agent that 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 differentiates itself in that the building blocks of our sequences are elementary behaviors instead of simple actions.

We have discussed at length the difference between pseudosequences and proper sequences, and we have shown that ALECSYS, our CS-based learning system, can learn proper sequences. (Pseudosequences were discussed in previous work; see Dorigo and Colombetti, 1994.)

An important aspect of our research is 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 that can cause a change to the next phase of the sequence: externally based transitions and result-based transitions. Each of these transition modalities corresponds to a training policy, based on the flexible RP and the rigid RP, respectively. These policies require the introduction of communication features into our system: trainer-to-agent communication, through a reinforcement sensor that makes explicitly available to the agent information about the quality of its behavior, and agent-to-trainer communication, through some observable behavior, to let the trainer know the current state of the agent. Most interestingly, the use of the reinforcement sensor introduces a new kind of rule, called *error recovery rules*, which are activated only in case of punishment. These rules tend to disappear as learning proceeds and performance improves.

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

Acknowledgments

This work has been partially supported by an Italian Ministry for University and for Scientific and Technological Research 60 percent grant for the year 1992 to Marco Colombetti and by a NATO-CNR Advanced Fellowship for the years 1993–1994 to Marco Dorigo.

The experiments were run by Sergio Barbesta, Jacopo Finocchi, and Maurizio Gorla. Helpful comments were made by Mukesh Patel, who read a previous version of the article, and by Jean-Arcady Meyer.

References

Beer, R. D. (1994). A dynamical systems perspective on autonomous agents. *Artificial Intelligence*, to appear.

Bertoni, A., & Dorigo, M. (1993). Implicit parallelism in genetic algorithms. *Artificial Intelligence*, *61*(2), 307–314.

Booker, L., Goldberg, D. E., & Holland, J. H. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, *40*(1–3), 235–282.

Colombetti, M., & Dorigo, M. (1992). Learning to control an autonomous robot by distributed genetic algorithms. In *Proceedings of "From Animals to Animats," Second International Conference on Simulation of Adaptive Behavior (SAB92)*. Cambridge, MA: MIT Press.

Dorigo, M. (1992). *ALECSYS and the AutonoMouse: Learning to control a real robot by distributed classifier systems* (Tech. Rep. No. 92-011). Milan, Italy: Politecnico di Milano.

Dorigo, M. (1993). Genetic and non-genetic operators in ALECSYS. *Evolutionary Computation, 1*(2), 151–164.

Dorigo, M., & Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, to appear. Also available as Tech. Rep. No. 92-040. Berkeley, CA: International Computer Science Institute.

Dorigo, M., & Schnepf, U. (1993). Genetics-based machine learning and behaviour-based robotics: A new synthesis. *IEEE Transactions on Systems, Man, and Cybernetics, 23*(1), 141–154.

Dorigo, M., & Sirtori, E. (1991). ALECSYS: A parallel laboratory for learning classifier systems. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. San Diego: Morgan Kaufmann.

Fitzpatrick, J. M., & Grefenstette, J. J. (1988). Genetic algorithms in noisy environments. *Machine Learning, 3*(2/3), 101–120.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley.

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

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*. Los Altos, CA: Morgan Kaufmann.

Lin, L.-J., & Mitchell, T. M. (1992). *Memory approaches to reinforcement learning in non-Markovian domains* (Tech. Rep. CMU-CS-92-138). Pittsburgh, PA: School of Computer Science, Carnegie Mellon University.

Littman, M. L. (1992). An optimization-based categorization of reinforcement learning environments. In *Proceedings of "From Animals to Animats," Second International Conference on Simulation of Adaptive Behavior (SAB92)*. Cambridge, MA: MIT Press.

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

McCluskey, E. J. (1986). *Logic design principles*. New York: Prentice-Hall.

Riolo, R. L. (1989). The emergence of coupled sequences of classifiers. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*. Los Altos, CA: Morgan Kaufmann.

Rosenschein, S. J., & Kaelbling, L. P. (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*. Los Altos, CA: Morgan Kaufmann.

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

Spiessens, P., & Manderick, B. (1991). A massively parallel genetic algorithm: Implementation and first analysis. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Los Altos, CA: Morgan Kaufmann.

Watkins, C. J. C. H. (1989). Learning with delayed rewards. Unpublished doctoral dissertation, University of Cambridge, England.

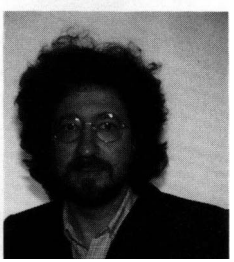
Watkins, C. J. C. H., & Dayan, P. (1992). Technical note: Q-learning. *Machine Learning*, *8*(3–4), 279–292.

Whitehead, S. D., & Lin, L. J. (1994). Reinforcement learning in non-Markov environments. *Artificial Intelligence*, to appear.

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

Wilson, S. (1990). The animat path to AI. In *Proceedings of "From Animals to Animats," First International Conference on the Simulation of Adaptive Behavior (SAB90)*. Cambridge, MA: MIT Press.

About the Authors



Marco Colombetti

Marco Colombetti received his Laurea (Master of Technology) in Electrical Engineering in 1976 from Politecnico di Milano, Italy. He is currently an Associate Professor at the Faculty of Engineering, Politecnico di Milano, Italy, where he gives a course on Knowledge Engineering. As a member of the Politecnico di Milano Artificial Intelligence and Robotics Project, he took part in several CEC ESPRIT Projects, and National research projects. He has been active in such diverse research fields as knowledge representation for cognitive modeling, plan-based models of human communication, and formal theories of mental states. His main current research interests concern the design, implementation, training and assessment of artificial agents with a learning component, and behavior engineering.

Marco Dorigo

Marco Dorigo received his Laurea (Master of Technology) in Industrial Technologies Engineering in 1986 and his Ph.D. in Electrical Engineering of Information and Systems in 1992 from Politecnico di Milano, Italy. He was a postdoctoral researcher at the International Computer Science Institute of Berkeley, CA, and he now holds a postdoctoral position at IRIDIA, Université Libre de Bruxelles, Belgium. He is a member of the Politecnico di Milano Artificial Intelligence and Robotics Project. He took part in several CEC ESPRIT projects and national research projects. His current research interests concern evolutionary computation, adaptive behavior, robotics, and behavior engineering.