Robot Shaping:
Developing Situated Agents through Learning*

Marco Dorigo#   Marco Colombetti+

INTERNATIONAL COMPUTER SCIENCE INSTITUTE

TR-92-040 Revised

April 1993

Abstract

Learning plays a vital role in the development of situated agents. In this paper, we explore the use of reinforcement learning to "shape" a robot to perform a predefined target behavior. We connect both simulated and real robots to ALECSYS, a parallel implementation of a learning classifier system with an extended genetic algorithm. After classifying different kinds of Animat-like behaviors, we explore the effects on learning of different types of agent's architecture (monolithic, flat and hierarchical) and of training strategies. In particular, hierarchical architecture requires the agent to learn how to coordinate basic learned responses. We show that the best results are achieved when both the agent's architecture and the training strategy match the structure of the behavior pattern to be learned. We report the results of a number of experiments carried out both in simulated and in real environments, and show that the results of simulations carry smoothly to real robots. While most of our experiments deal with simple reactive behavior, in one of them we demonstrate the use of a simple and general memory mechanism. As a whole, our experimental activity demonstrates that classifier systems with genetic algorithms can be practically employed to develop autonomous agents.

* This work has been submitted to the Artificial Intelligence Journal and 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 about learning, in two different senses. It is about an automatic learning system used to develop behavioral patterns in an autonomous agent, a simple mouse-like robot that we call the AutonoMouse. Moreover, it is about what *we* learned on designing and training autonomous agents to act in the world.

Broadly speaking, our work situates itself in the recent line of research which concentrates on the realization of artificial agents strongly coupled with the physical world, and usually dubbed *embedded* or *situated agents*. Paradigmatic examples of this trend are the works by Agre & Chapman (1987), Kaelbling (1987), Brooks (1990a, 1991a), Kaelbling & Rosenschein (1991), Whitehead & Ballard (1991), and others. While there are important differences among the various approaches, some common points seem to be well established. A first, fundamental requirement is that agents must be grounded, in that they must be able to carry on their activity in the real world and in real time. Another important point is that adaptive behavior cannot be considered as a product of an agent considered in isolation from the world, but can only emerge from a strong coupling of the agent and its environment.

There are basically two ways to obtain such a coupling. The first way relies on smart design: the agent 's designer analyzes the dynamics of the complex system made up by the agent and the environment, so that such dynamics can be exploited to produce the desired interactions. This approach has been pioneered by Rosenschein & Kaelbling (1986). More recently, Agre & Horswill (this volume) have focused their attention on the aspects of the environment that make complex action without prior planning possible; Horswill (this volume) is studying so called *habitat constraints,* which define the set of environments in which an agent can operate; and Hammond, Converse & Grass (this volume) are studying how an agent can actively *stabilize* the environment to make it more hospitable.

The second approach relies on automatic learning to dynamically develop a situated agent through interaction with the world. The idea is that the interactions between an agent and its environment soon become very complex, and their analysis is likely to be a hard task. Moreover, the classical design method based on the factorization of a complex system into a network of modular subsystems is likely to constrain the space of possible designs in such a way that many interesting, nonmodular solutions will be excluded (Beer, this volume).

The approach we advocate is intermediate. First, we design the learning system architecture in such a way to favor learning basing our design choices on a detailed analysis of the task and of the interactions between the agent and the world; in this phase smart design will exploit the environment's characteristics in order to make learning possible.

Second, we use learning as a means to translate suggestions coming from an external trainer into an effective control strategy that allows the agent to achieve a goal; this kind of *supervised* reinforcement learning scheme has been applied to real robots by Mahadevan & Connell (1992) and by us. We call this approach *shaping*, as opposed to the more classical *unsupervised* reinforcement learning approach, in which an organism increasingly adapts to its environment by directly experiencing the effects of its activity (in this volume this approach is discussed by Barto, Bradtke & Singh, and by Whitehead & Lin).

The problem we face is therefore to find a right balance between design, learning and training, that is between the knowledge we craft into the agent and the knowledge the agent is to find out by interaction with the environment under the guidance of the trainer. To solve this problem we rely heavily on experimentation, in that different design choices and different training and learning strategies must be compared through experimental activity. We therefore ran many experiments with both simulated agents and real robots. These experiments are discussed in the paper, that is organized as follows. In Section 2 we describe the agents, environments and behavioral patterns we have used in our experiments. Section 3 summarizes the reinforcement learning technique we have used and illustrates ALECSYS, the software tool we have developed to implement learning agents. Section 4 provides a characterization of those features of the environment that allow a trainer to steer our agents toward the desired patterns of interaction. In Sections 5 we discuss different kinds of architecture and learning strategies that can be used to implement the agent's behavior. Sections 6 and 7 present some experiments carried out by simulation and in the real world. In Section 8 we survey related work. Finally, in Section 9 we draw some conclusions and suggest directions for further research.

2. The AutonoMouse and its world

Behavior is a product of the interaction between an agent and its environment. The universe of possible behavioral patterns is therefore determined by the structure and the dynamics of both the agent and the environment, and by the interface between the two (the sensors and the effectors). In this section, we describe the agents, the environments and the behavioral patterns we have chosen to carry out our experiments.

The agent's anatomy

Our artificial agent, the AutonoMouse, is a small moving robot. So far, we have experimented with two versions of it, that we call AutonoMouse II and AutonoMouse IV, respectively described in Figures 1 and 2. Pictures of AutonoMouse II and of AutonoMouse IV are presented respectively in Figures 3a and 3b.

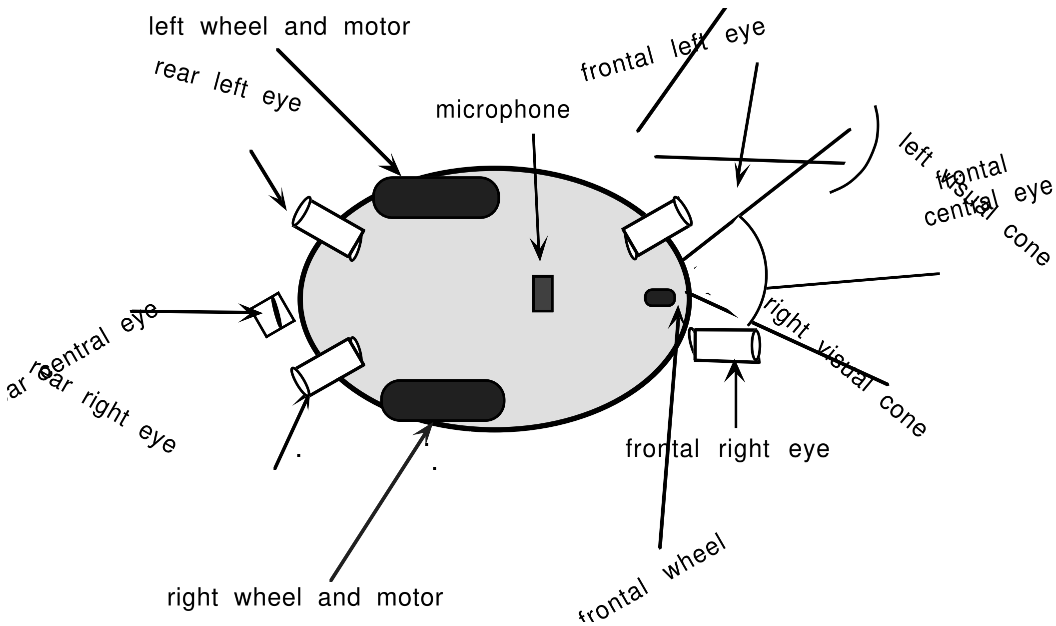


Figure 1. *Description of AutonoMouse II*.

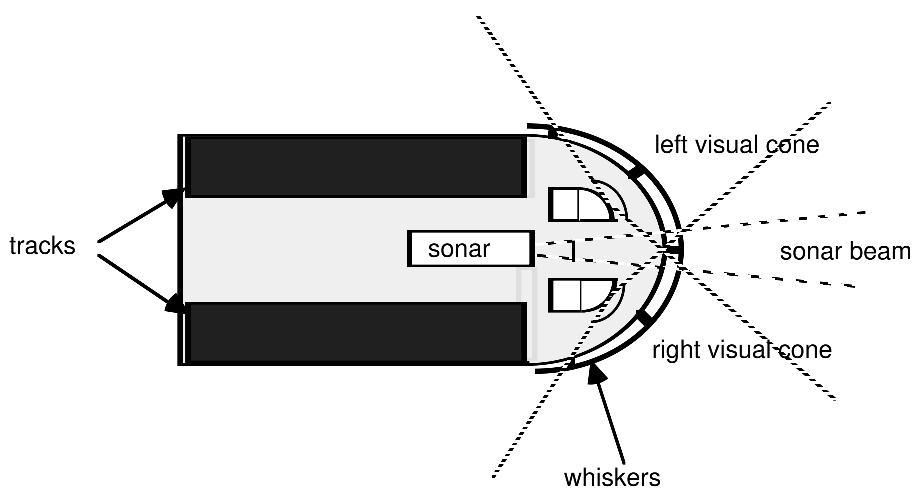


Figure 2. *Description of AutonoMouse IV.*

AutonoMouse II has four directional eyes and two motors. Each directional eye can sense a light source within a cone of about 60 degrees. Each motor can stay still or move the connected wheel one or two steps forwards, or one step backwards. AutonoMouse II is connected to a transputer board on a PC via a 9600-baud RS-232 link. Only a small amount of processing is done on-board (the collection of data from sensors and to actuators and the management of communications with the PC). All the learning algorithms run on the transputer board.

AutonoMouse IV has two directional eyes, a sonar, front and side whiskers, and two motors. Each directional eye can sense a light source within a cone of about 180 degrees. The two eyes together cover a 270 degrees zone, with an overlapping of 90 degrees in front of the robot. The sonar is highly directional and can sense an object as far as 10 meters. For the purposes of the experiment presented in Section 7 the output of the sonar can assume two values, either *I_sense_an_object*, or *I_do_not_sense_an_object*. Each motor can stay still or move the connected track one or two steps forwards, or one step backwards. AutonoMouse IV is connected to a transputer board on a PC via a 4800-baud infra-red link.

The simulated AutonoMice are basically the models of their physical counterparts.

a) b)

Figure 3. *a) AutonoMouse II's portrait,*
b) AutonoMouse IV's portrait.

The agent's "mind"

The AutonoMouse is connected to ALECSYS (A LEarning Classifier SYStem), a classifier system with a genetic algorithm implemented on a network of transputers (Dorigo & Sirtori, 1991). We chose to work with learning classifier systems because they seem particularly fit to implement simple reactive interactions in an efficient way; still, their use leaves open the possibility to study, in future extensions of our work, issues arising from delayed reinforcement.

The environment

We would like our environment to be inhabited by such things as preys, sexual partners, predators, etc. More modestly, the AutonoMouse is presently able to deal reasonably well with much poorer entities, like slowly moving lights, steady obstacles, and sounds. Of course, we could fantasize freely in simulations, by introducing virtual sensors able to detect the desired entities, but then results would not carry to real experimentation; so, we prefer to adapt our goals to the actual capabilities of the agent.

Behavior

A first, rough classification allows one to distinguish between *Stimulus-Response (S-R) behavior*, i.e. reactive responses connecting sensors to effectors in a direct way, and *dynamic behavior*, requiring some kind of internal state to mediate between input and output. Although in some experiments we have built rudimentary kinds of dynamic behavior, so far we have been mainly working with S-R responses.

In our work we have been influenced by Wilson's Animat problem (1987), that is the issue of realizing an artificial system able to adapt and survive in a natural environment. This means that we are interested in behavioral patterns that are the artificial counterparts of basic natural responses, like feeding and escaping from predators. Our experiments are therefore to be seen as possible solutions to fragments of the Animat problem.

We believe that experiments on situated agents must be carried out in the real world to be truly significant. However, such experiments are in general costly and time-consuming. It is therefore advisable to preselect a small number of potentially relevant experiments to be performed in the real world. To carry out the selection we use a simulated environment, which allows us to have accurate expectations on the behavior of the real agent and to prune the set of possible experiments.

One of the hypotheses we want to explore is that relatively complex behavioral patterns can be built bottom-up from a set of simple responses. This hypothesis has already been put to test in robotics, for example by Arkin (1990) with his Autonomous Robot Architecture that integrates different kinds of information (perceptual data, behavioral schemes and world knowledge) in order to get a robot to act in a complex natural environment. Arkin's robot generates complex responses, like walking through a doorway, as a combination of competing simpler responses, like moving ahead and avoiding a static obstacle (the wall, in the doorway example). The key point is that complex behavior can demonstrably emerge from the simultaneous production of simpler responses. We have considered five kinds of basic responses:

- The *approaching behavior,* i.e. getting closer to an almost still object with given features; in the natural world, this response is a fundamental component of feeding and sexual behavior.
- The *chasing behavior*, i.e. following and trying to catch a moving object with given features; as the preceding approaching behavior, this response is important for feeding and reproduction.
- The *mimetic behavior*, i.e. entering a well-defined physical state which is a function of a feature of the environment; this is inspired by the natural behavior of a chameleon, changing its color according to the color of the environment.
- The *avoidance behavior*, i.e. avoiding physical contact with an object of a given kind; this can be seen as the artificial counterpart of a behavioral pattern which allows an organisms to avoid hurting objects.
- The *escaping behavior*, i.e. moving as far as possible from an object with given features; the object can be viewed as a predator.

More complex behavioral patterns can be built from these simple responses in many different ways. So far, we have studied the following building mechanisms:

- *Independent sum*: two or more independent responses are produced at the same time; for example, an agent may assume a mimetic color while chasing a prey.
- *Combination*: two or more homogeneous responses are combined into a resulting behavior; consider the movement of an agent following a prey and trying to avoid an obstacle at the same time.
- *Suppression*: a response suppresses a competing one; for example, the agent may give up chasing a prey in order to escape from a predator.
- *Sequence*: a behavioral pattern is built as a sequence of simpler responses; for example, fetching an object involves reaching the object, grasping it, and coming back.

In general, more than one mechanism can be at work at the same time: for example, an agent could try to avoid still hurting objects while chasing a moving prey and being ready to escape if a predator is perceived.

The trainer

Training an agent means making its behavior converge to a predefined *target behavior*. While this is the case for any learning scheme allowing for supervised learning, the way in which the trainer can exert her supervision varies from scheme to scheme. For example, most learning schemes used with neural networks require comparing the network's actual response with the "correct" response, as predefined by the trainer. This scheme is not fit for training a real robot, though, because the correct behavior cannot easily be presented for a comparison. Instead, we have adopted a reinforcement scheme, i.e. a learning mechanism able to accept from the trainer a positive or negative reinforcement as a consequence of a response.

In the literature, the term "reinforcement learning" mostly refers to unsupervised learning contexts: an agent interacts with its environment in a completely unsupervised setting, and receives a reward only when it achieves a final goal. This setting closely resembles a natural situation, in which an organism is only occasionally rewarded by its environment. It seems to us, however, that this kind of unsupervised learning alone is not suitable to develop effective robots. In fact, unsupervised learning provides little useful information to the agent, and this results into very slow learning rates. Contrary to natural situations, in artificial settings we can have a trainer at our disposal, and there is no reason not to exploit her knowledge to achieve faster learning.

Training an artificial robot closely resembles what experimental psychologist do in their laboratories, when they train an experimental subject to produce a predefined response. To stress this similarity, we have borrowed the term *shaping* from experimental psychology (this term dates back at least to Skinner, 1938, and has already been used in machine learning by Singh, 1992). It turns out that our trainer is similar to what Whitehead (1991a; 1991b) calls *external critic*. A similar method has already been proved to be effective by Mahadevan & Connell (1992).

A shaping setting includes an agent, an environment, and a trainer. In principle, the trainer could be a human being observing the agent's interaction with the environment, and issuing reinforcements consequently; for efficiency reasons, however, reinforcements are provided automatically by a *reinforcement program* (RP).

The role of the RP in shaping the robot's behavior is critical, in that it embodies the trainer's characterization of the target behavior. If we compare robot shaping with traditional task-level robot programming, the RP can be viewed as a sort of source code which has to be translated into the robot's control program. The learning mechanism plays the role of a *situated translator* – that is, a translator which is sensitive to the actual interaction between the agent and the world. And it is precisely through the world sensitivity of learning that a proper degree of situatedness can be achieved.

3. The learning system

Here we briefly illustrate some characteristics of ALECSYS, a parallel learning classifier system allowing for the implementation of hierarchies of classifier systems, which can be exploited to build modular agents.

ALECSYS introduces some major improvements in the standard model of learning classifier systems (CS) (Booker, Goldberg & Holland, 1989). First, ALECSYS permits to distribute a CS on any number of transputers (Dorigo & Sirtori, 1991; Dorigo, 1992a, 1992c) . Second, it gives the learning system designer the possibility to use many concurrent CSs, each one specialized in learning a specific behavioral pattern. Using this feature the system designer can use a divide-and-conquer approach: the overall learning task is decomposed in several learning subtasks (easier and quicker to learn), which are coordinated by *coordination modules* which are themselves learning subtasks[1]. Our agents are therefore not completely built through learning; they also have a certain amount of "innate" architecture. (Innate architecture is created by the way in which the global system is built from interconnected classifier subsystems.) Third, ALECSYS introduces a set of new operators that overcome some of the problems and inefficiencies of previous CS implementations. This last point will not be considered here; details about the algorithms can be found in Dorigo (1993). In our experiments we used an enhanced version of the basic algorithm presented in the next subsection.

The learning classifier system paradigm

As the model proposed by Booker, Goldberg & Holland (1989), our learning classifier systems are composed of three main components (see Figure 4).

- The *performance* module, which is a kind of parallel production system, implementing a behavioral pattern as a set of condition-action rules, or *classifiers*. Our classifiers have two conditions and one action. Conditions and actions are strings of fixed length k; symbols in the condition string belong to {0,1,#}, symbols in the action string belong to {0,1}.
- The *credit apportionment* module, which is responsible for the redistribution of incoming reinforcements to classifiers. Basically, the algorithm is an extended version of the *bucket brigade* described by Dorigo (1993).

---

1 This technique is somewhat reminiscent of the approach taken by Mahadevan & Connell (1992). The main difference is that we not only learn basic behaviors, but we also learn how to make them interact (i.e., their coordination); in the work of Mahadevan & Connell, coordination is achieved by a hard-wired subsumption architecture. Another difference is that we use learning classifier systems instead of Q-learning with statistical clustering.

- The *rule discovery* module, which creates new classifiers according to an extended genetic algorithm (Dorigo, 1993).

Learning takes place at two distinct levels. First, the apportionment of credit can be viewed as a way of learning from experience the adaptive value of a number of given classifiers with respect to a predefined target behavior. Second, the rule discovery mechanism allows the agent to explore the value of new classifiers.

In CSs the *bucket brigade* algorithm solves both the structural and temporal credit assignment problems (see for example Sutton, 1988). Every classifier maintains a value, called *strength*, that is modified by the bucket brigade in an attempt to redistribute rewards to classifiers that are useful and punishments to those that are useless (or harmful). Strength is used to assess the degree of usefulness of classifiers; classifiers that have all conditions satisfied are fired with a probability that is a function of their strength. The genetic algorithm explores the classifiers space recombining useful classifiers to produce possibly better offspring. Offspring are then evaluated by the bucket brigade.

An example can help to understand how the CS model works (see Figure 4). Consider AutonoMouse II (Figures 1 and 3a) and the learning task *approaching a light source*. The learning system is initialized by a set of randomly generated classifiers, each with the same strength. The CS receives 4-bit input messages, identifying the light position (see below and Figure 5 for details), which are appended to the message list, a data structure initially empty. Messages in the message list are then matched against conditions of classifiers; matching classifiers are activated for inclusion in the next stage. The auction module chooses probabilistically within the set of activated classifiers those which are allowed to append a message to the message list. (A classifier has a probability to win the auction proportional to its strength.) Some of the messages appended can be sent to effectors: they are proposing actions (e.g., robot moves). If the proposed actions are not conflicting, then the actions are carried out. Otherwise a conflict resolution mechanism is called. The conflict resolution mechanism could, for example, choose one of the conflicting actions probabilistically, with a probability proportional to the strength of the classifier that proposed the action. This action is rewarded (or punished) by the trainer.

As the classifier set is randomly generated, with high probability it does not contain all the rules necessary to accomplish satisfactorily the task. It is the duty of the genetic algorithm to recombine classifiers and to substitute low strength ones with new ones. The genetic algorithm (Holland, 1975) will not be discussed here as it is a well-established algorithm.

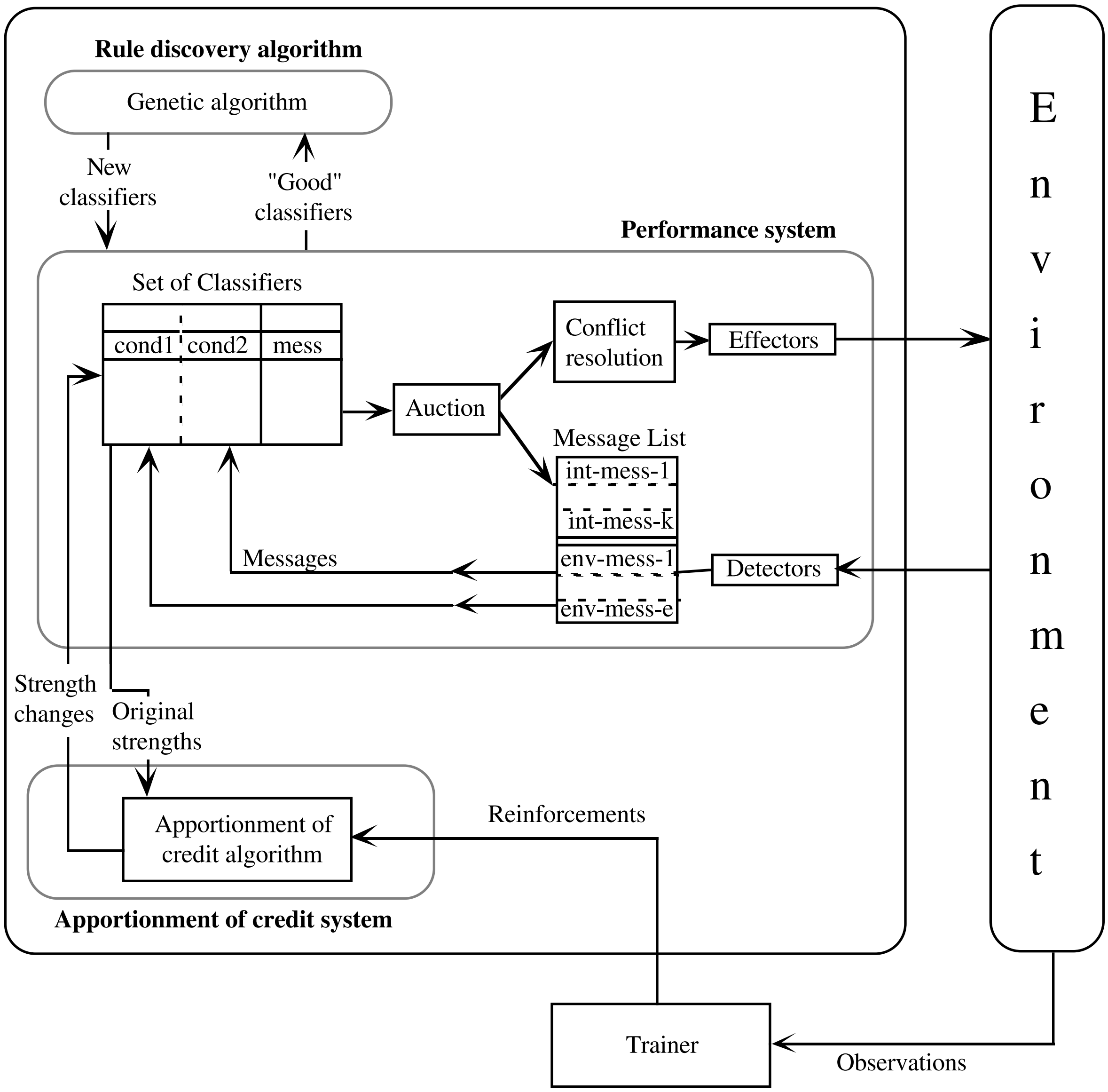


Figure 4. *The learning classifier system.*

Basic and coordination behaviors in ALECSYS

With ALECSYS it is possible to define two classes of learning modules; we call them *basic behaviors* and *coordination behaviors.* Both are implemented as classifier systems.

Basic behaviors are directly interfaced with the environment. Each basic behavior receives bit strings as input from sensors and sends bit strings to actuators to propose actions. Basic behaviors inserted in a hierarchical architecture occupy level 1; they send bit-strings to connected higher level coordination modules. Consider for example AutonoMouse II and the basic behavioral pattern Chase. As all behaviors (both basic and coordination ones), it is implemented as a CS. For ease of reference we call this classifier system CS-Chase. Figure 5 shows the input-output interface of CS-Chase. In this case the input pattern only says which sensors see the predator. (AutonoMouse II has four binary sensors, see Figures 1 and 3a, which are set to 1 if light intensity is higher than a given threshold, to 0 otherwise.) The output pattern is composed of a proposed action, a *direction of motion* plus a *move/do_not_move* command, and of a bit string (in this case of length 1) for the coordinator; this bit-string is there to let the coordinator know that CS-Chase was proposing an action. Note that the value of this bit string is not designed, but must also be learned by CS-Chase.

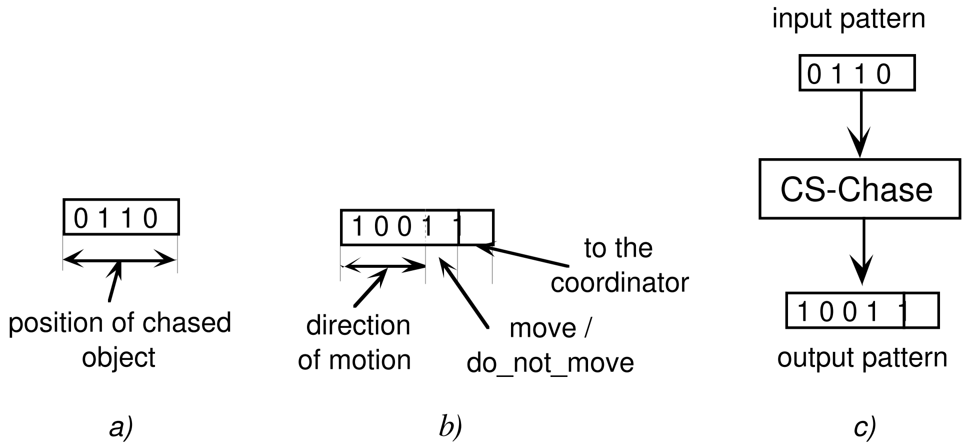


Figure 5. a) *Example of input message,*
b) *Example of output message,*
c) *Example of input-output interface for the CS-Chase behavior.*

Coordination behaviors receive input from lower level behavioral modules and produce an output action that, with different modalities depending on the composition rule used, influences the degree of application of actions proposed by basic behaviors. Figure 6 shows one possible innate architecture of an agent that has the following learning task (which we call the Chase/Feed/Escape behavior):

```
If  there is a predator
  then  Escape
  else  if  hungry
          then  Feed {i.e., search for food}
          else  Chase the moving object.
```

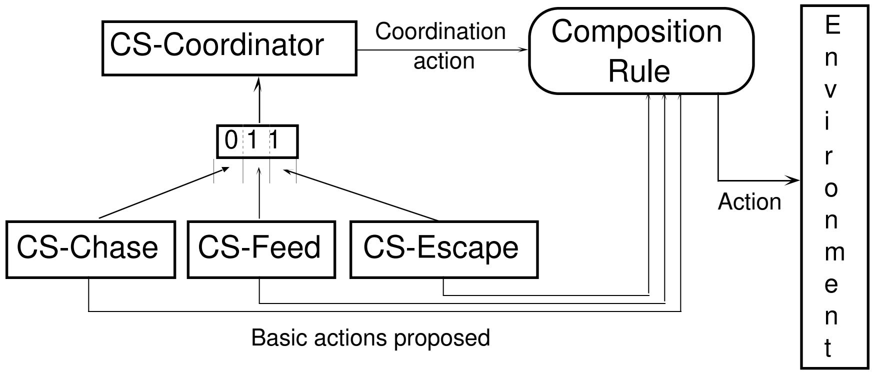


Figure 6. *Example of innate architecture for a three behaviors learning task.*

In our simulated environment predators appear at random time intervals; the agent becomes hungry whenever it sees a food source; the moving object is always present (this means that at least one basic behavioral module is always active).

In this example, a basic behavior has been designed for each of the three behavioral pattern used to describe the learning task. In order to coordinate basic behaviors in situations in which two or more of them propose actions simultaneously, a coordination module is used. It receives a bit string from each connected basic behavior (in this case a one-bit string, the bit indicating whether the sending CS wants to do something or not) and proposes a *coordination action*. This coordination action goes into the composition rule module, which implements the composition mechanism. In this example the composition rule used is *suppression*, and therefore only one of the basic actions proposed is applied.

4. Interdependence between the environment, the learning agent, and the trainer

Our scenario includes an environment, a learning agent, and a trainer in charge of shaping agent/environment interactions. Even if our agents and environments are very simple, to characterize their interactions is by no means trivial. First, the agent's architecture is not given *a priori*, but is at least partially designed in order to fit a given situation. Also the environment is not completely "natural", in that it contains artificial objects that can be exploited in order to make the intended interactions possible. Moreover, there are many different ways in which one may attempt to shape the agent's behavior.

In general, we start with some intuitive idea of a target behavior in mind. We consider whether the natural characteristics of the environment are likely to suit such behavior, or whether we need to enrich the environment with appropriate artificial objects, like moving lights and special surfaces. Then we design a sensorimotor interface and an internal architecture that allows the agent to gather enough information from the environment, and to act back on the environment so that the desired interaction can emerge. Finally, we ask ourselves what *shaping policy* (i.e., strategy in providing reinforcements) can actually steer the agent toward the target behavior. This process is iterative, in that difficulties in finding, say, an appropriate shaping policy may compel us to backtrack and modify previous design decisions.

In the following, we discuss the relevant aspects of all entities involved in making a pattern of interaction emerge.

Properties of actions

Consider the five basic responses introduced in Section 2. Four of them are *objectual*, in that they involve the agent's relationship with an external object; these responses are the approaching, chasing, avoidance, and escaping behaviors. One response, namely the mimetic behavior, is not objectual, in that it involves only states of the agent's body.

Objectual responses are:

- *type-sensitive*, in that agent/object interactions are sensitive to the type to which the object belongs (prey, obstacle, predator, etc.);
- *location-sensitive*, in that agent/object interactions are sensitive to the relative location of the object with respect to the agent.

Type-sensitivity is interesting because it allows for fairly complex patterns of interaction, which are however within the capacity of an S-R agent. In fact, it requires only that the agent be able to discriminate some object feature characteristic of the type. Clearly, the types of objects an S-R agent can tell apart depend on the physical interactions between external objects and the agent's sensory apparatus. Note that an S-R agent is not able to *identify* an object, which means discerning two identical but distinct objects of the same type.

The interactions we consider do not depend on the absolute location of the objects and of the agent; in fact, they depend only on the relative angular position, and sometimes on the relative distance, of the object with respect to the agent. Again, this requirement is within the capacities of an S-R agent.

It is important to note that an agent's behavior can only be understood in relation with the environment. For example, the difference between the avoidance behavior and the escaping behavior cannot be understood by considering the agent in isolation from its environment. In both behaviors, the agent's task is just to increase the distance between itself and some external object. However, an external observer understands the agent to avoid *obstacles* (i.e., still or at most "blindly" moving objects), while she understands the agent to escape from *predators* (i.e., objects that may actively try to chase it).

In the context of shaping, differences that appear to an external observer can be relevant even if they are not perceived by the agent. The reason is that the trainer will in general base her reinforcing activity on an observation of the agent's interaction with the environment, and not on the agent's internal states alone. Clearly, from the point of view of the agent a single move of the avoidance or of the escaping behavior are exactly the same. However, in complex behavior patterns, avoidance and escaping relate differently to other behaviors. In general, avoidance should modulate some other movement response; on the contrary, escaping will be more successful if it suppresses all competing responses. As we shall see in the following sections, this fact is going to influence both the architectural design and the shaping policy for the agent.

Properties of the environment

For learning to be successful, the environment must have a number of properties. Given the kind of agent we have in mind, the interaction of a physical object with the agent depends only on the object's type and on its relative position with respect to the agent. Therefore, sufficient information about object types and relative positions must be available to the agent. This problem can be solved in two ways: either the "natural" objects existing in the environment have sufficient distinctive features that allow them to be identified and located by the agent, or else "artificial" objects must be designed so that they can be identified and located. For example, if we want the agent to approach light L1 and avoid light L2, the two lights must be of different color, or have a different polarization plane, to be distinguished by appropriate sensors. In any case, identification will be possible only if the rest of the environment cooperates. For example, if light sensing is involved, environmental lighting must be almost constant during the agent's life.

In order for a suitable response to depend on an object's position, objects must be still, or move slowly enough with respect to the agent's speed (this aspect will be further discussed below). This does not mean that a sufficiently smart agent could not evolve a successful interaction pattern with very fast objects: however, such a pattern could not depend on the instantaneous relative position of the object, but would involve some kind of extrapolation of the object's trajectory, which is beyond the present capacities of the AutonoMice.

Properties of the learning system

The learning system we use is based on the metaphor of biological evolution. This raises the question of whether evolution theory provides the right technical language to characterize the learning process.

We think we should resist this temptation. There are various reasons why the language of evolution cannot literally apply to our agents. First, we use an evolutionary mechanism to implement individual learning rather than philogenetic evolution. Second, the distinction between phenotype and genotype, which is essential in evolution theory, in our case is rather confused; in fact, individual rules within a CS play both the role of a single chromosome and of the phenotype undergoing natural selection. In our experiments, we found that we tend to consider the learning system as a black box, able to produce S-R associations and categorizations of stimuli into relevant equivalence classes. More precisely, we expect the learning system:

- to discover useful associations between sensory input and responses;
- to categorize input stimuli so that precisely those categories will emerge which are relevantly associated to responses.

Given these assumptions, the sole preoccupation of the designer is that the interactions between the agent and the environment can produce enough relevant information for the target behavior to emerge. As it will appear from the experiments reported in the following sections, this concern influences the design of artificial environment objects and of the agent's sensory interface.

The trainer as an agent

In principle, the trainer is an agent, with own sensors, effectors and control. The trainer's sensors allow her to observe the behavior of the robot to be shaped, her effectors are used to provide reinforcements, and her control system implements a given shaping policy. Note that the trainer's environment includes both the robot's environment and the robot itself.

As we have already said, in the experiments reported in this paper the role of the trainer is played by the reinforcement program (RP). For the implementation of the RP, the only nontrivial function is the observation of the agent's behavior. In fact, previous research in robot shaping has solved this problem by identifying the RP's sensors with the agent's sensors, i.e. by providing the trainer exactly with the same input information that is fed into the robot (see Mahadevan & Connell, 1992). This approach has some shortcomings. First, it does not allow the trainer to gather more information about the environment than the agent does, which seems to be an unnecessary limitation. Second, and more important, it binds the shaping policy to depend on low-level details of the agent's physical structure. As a consequence, the RP will in general be as complex as a program directly implementing the target behavior, and this greatly limits the effectiveness of learning as an alternative to robot programming; moreover, any low-level change to the agent's physical architecture makes it necessary to write a new RP.

In our opinion, RPs should be easier to write than control programs, and should be portable from agent to agent, at least when the differences are not too large. To achieve this result, an RP must be abstract enough and independent of the agent's internal structure. Often, this involves providing the RP with own sensors, able to extract information from the environment independently of the agent.

To give a concrete example, in the experiments with AutonoMouse II (see Section 7), the robot used only binary information from its four directional eyes, while the RP used the two central eyes (Figure 1) placed on the robot to evaluate the increase or decrease of light intensity, which is related to the distance from the light source. In other words, the robot carried the trainer's sensors on board. In the experiment with AutonoMouse IV (also reported in Section 7) we have followed a different strategy: the same hardware devices are used both as the sensors of the agent and as the sensors of the RP. However, while the 8-bit outputs of such devices are used directly by the RP, they are transformed into simpler on/off signals before being input to the robot. In this way, the agent receives enough information to implement the target behavior, but its learning speed profits from the reduction of the search space size.

As a consequence of these design decisions, the very same RP can be used to shape a variety of different agents, provided their sensory apparatus is fine enough to support the relevant discriminations in the given environment. The conceptual analysis of the target behavior necessary for writing the RP can be highly independent of the agent to be shaped, thus making the RP portable from agent to agent. This is coherent with our claim that reinforcement learning can be seen as a kind of situated translation of a high level specification of the target behavior (see end of Section 2). The learning mechanism, regarded as a translator, is *machine independent* in that it need not embed a model of the device for which the control program is produced. And the trainer, regarded as a robot programmer, can concentrate on her own view of the interaction, neglecting the agent's architecture as far as the agent is sufficiently powerful to discriminate relevant world states.

Beyond reactive behavior

In one of our experiments, we tried to go beyond simple S-R behavior. As remarked by Beer (this volume), this implies that the agent is endowed with some form of internal state (which need not be regarded as a "representation" of anything). The most obvious candidate for an internal state is a memory of the agent's past (Whitehead & Lin, this volume). Of course, the designer has to decide *what* has to be remembered, *how* to remember it, and for *how long*. Such decisions cannot be taken without a prior understanding of relevant properties of the environment.

In an experiment reported in Section 6, we added a memory of the past state of the agent's sensors, allowing the learning system to exploit regularities of the environment. The idea is that if physical objects are still or move slowly with respect to the agent, their current position is strongly correlated with their previous position. Therefore, how an object was sensed in the past is relevant to the actions to be performed *now*, even if the object is not currently perceived.

In fact, suppose that at cycle N the agent senses a light in the *leftmost area* of its visual field, and that at cycle N+1 the light is no more sensed. This piece of information is useful to approach the light, because at cycle N+1 the light is likely to be out of the agent's visual field *on its left*. The experiments showed that a memory of past perceptions initially makes the learning process harder, but eventually increases the performance of the approaching behavior.

By running a number of such experiments, we confirmed an obvious expectation, i.e. that the memory of past perceptions is useful only if the relationship between the agent and its environment changes slowly enough to preserve a high correlation between subsequent states. In other words, agents with memory are favored only in reasonably predictable environments.

Learning versus design

As we have already remarked, successful learning presupposes a careful design of the agent's interface, and possibly of artificial world objects. A further design issue regards the controller's architecture, i.e. the overall structure of the system in charge of producing actual behavior. This issue is particularly relevant when the target behavior is not a basic response, but a complex behavior pattern.

In principle, also complex behavior patterns, like the ones presented in Section 2, can be learned by a single classifier system. However, learning might be very slow, because more complex behaviors correspond to larger search spaces for both credit apportionment and rule discovery. It is therefore interesting to see whether a search space can be factored into a number of smaller spaces. This question brings in the issue of architecture: intuitively, when a complex behavior pattern can be decomposed into simpler elements, some kind of hierarchical architecture is expected to speed up learning as a result of narrowing search. In fact, the use of a prewired architecture is also suggested by results obtained by other researchers in the field of autonomous systems (e.g., Mahadevan & Connell, 1992; Mahadevan, 1992),

As we shall see in Sections 6 and 7, the experiments carried out to systematically compare different types of architectures confirm this expectation. Different kinds of complex behavior do profit from different types of architectures; at the same time, each type of architecture constrains the shaping procedure, that is the strategy adopted to drive learning. These issues are dealt with in the next section.

5. Types of architectures and shaping policies

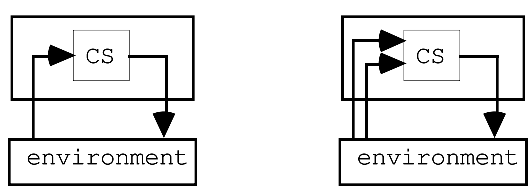
In ALECSYS, an agent can be implemented by a network of different CSs. The issue of architecture is therefore the problem of designing the network that best fits some predefined class of behaviors. So far, we have experimented with different types of architectures, that can be broadly classified in two classes:

- *monolithic architectures*, built by one CS directly connected to the agent's sensors;
- *distributed architectures*, built by many CSs; in this case we distinguish between two subclasses:
  - *flat architectures*, built by more than one CS, in which all CSs are at "level 1", i.e. directly connected to the agent's sensors;
  - *hierarchical architectures*, built by a hierarchy of levels.

Within such classes, there are still a number of possible choices, as described below.

Monolithic architectures

The simplest choice is, of course, the *monolithic architecture*, with only one CS in charge of controlling the whole behavior[2] (Figure 7). If the target behavior is made up of several basic responses, there is a further choice to be made: the state of all sensors can be wrapped up in a single message (Figure 7a), or distributed into a set of independent messages (Figure 7b). We call the latter case *monolithic architecture with distributed input*. The idea is that inputs relevant to different responses can go into distinct messages; in such a way, input messages are shorter, and the overall learning effort can be reduced (see the "Monolithic architecture with distributed input" experiment in Section 6).



a *b*

Figure 7. *Monolithic architectures.*

Flat architectures

A distributed architectures is made up of more than one CS. If all CSs are directly connected to the agent's sensors, then we use the term *flat architecture* (Figure 8). The idea is that distinct CSs implement the different basic responses that make up a complex behavior pattern. There is a

[2] Mahadevan & Connell (1992) first proposed the term *monolithic architecture* for this kind of structure.

further issue, here, regarding the way in which the agent's response is built up from the moves proposed by the distinct CSs. If such moves are independent, they can be realized by different effectors at the same time (Figure 8a); those moves that are non independent, however, have to be integrated into a single response before they are realized (Figure 8b).

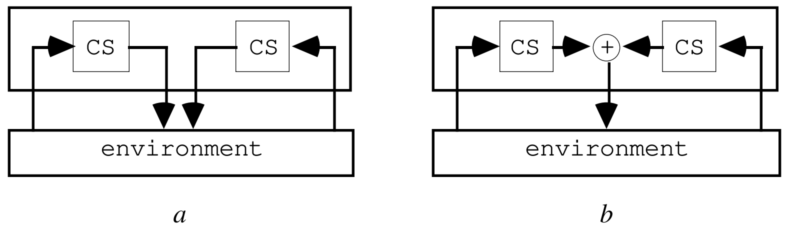


Figure 8. *Flat architectures.*

Hierarchical architectures

In a flat architecture, all CSs receive input only from the sensors. In a *hierarchical architecture*, the set of all CSs can be partitioned into a number of *levels*. By definition, a CS belongs to level N if it receives input from systems of level N–1 at most, where level 0 is defined as the level of sensors. An N-level hierarchical architecture is a hierarchy of CSs having level N as the highest one; Figure 9 shows two different 2-level hierarchical architectures. First level CSs implement basic behaviors described in Section 3, higher level CSs implement coordination behaviors.

With a CS in a hierarchical architecture we have two problems: first, how to receive input from a lower level CS; second, what to do with the output. Receiving input from a lower level CS is easy: remember that all messages are bit strings of some fixed length; therefore, an output message produced by system CS_1 can be treated as an input message by a different system CS_2. In a sense, lower-level CSs are viewed by higher-level ones as virtual sensors.

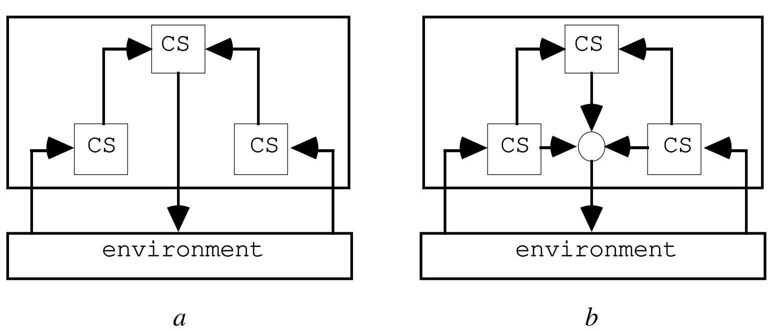


Figure 9. *Two-level hierarchical architectures.*

The problem of deciding what to do with the output of CSs is more complex. In general, the output messages from the lower levels go to higher-level CSs, while the output messages from the higher levels can go directly to the effectors to produce the response (Figure 9a), or be used to control the composition of responses proposed by lower CSs (Figure 9b). In this paper, most of the experiments were carried out using "suppression" as composition rule; we dub the resulting hierarchical systems *switch architectures.* In Figure 10 we show an example of three-level switch architecture implementing an agent which should learn the Chase/Feed/Escape behavior introduced in Section 3. In this example the coordinator of level two (SW1) should learn to suppress the Chase behavior whenever the Feed behavior proposes an action, while the coordinator of level three (SW2) should learn to suppress SW1 whenever the Escape behavior proposes an action.

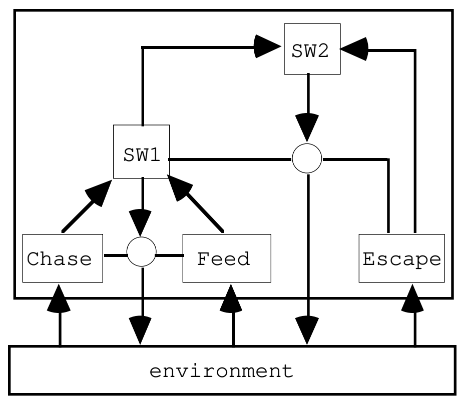


Figure 10. *An example of three-level switch architecture for the Chase/Feed/Escape behavior. Besides the three basic behaviors can be seen the two switches, SW1 and SW2.*

How to design an architecture: Qualitative criteria

The most general criterion for choosing an architecture is to make the architecture naturally match the structure of the target behavior. This means that each basic response should be assigned a CS, and that such CSs should be connected in the most natural way to obtain the global behavior.

Suppose the agent should normally follow a light, while being ready to reach its nest if a specific noise is sensed (revealing the presence of a predator). This behavior pattern is made up of two basic responses, namely following a light and reaching the nest, and the relationship between the two is one of suppression (see Section 2). In such a case, the switch architecture is a natural choice.

In general, the four mechanisms for building complex behaviors defined in Section 2 map onto different types of architecture in the following way:

- *Independent sum*: flat architecture with independent outputs (Figure 8a).
- *Combination*: flat architecture with integrated outputs (Figure 8b), or hierarchical architecture.
- *Suppression*: switch architecture (remember that the switch architecture is a special kind of hierarchical architecture).
- *Sequence* (not treated in this paper, see Colombetti & Dorigo, 1993): hierarchical architecture.

How to design an architecture: Quantitative criteria

In Section 4 we stressed that the main reason for introducing architecture is speeding up learning of complex behavior patterns. Clearly, speed-up is the result of factoring a large search space into smaller ones; therefore, a distributed architecture will be useful only if the component CSs have smaller search spaces than a single CS able to perform the same task.

We can turn this consideration into a quantitative criterion, by observing that the size of a search space grows exponentially with the length of messages. This implies that a hierarchical architecture can be useful only if the lower-level CSs realize some kind of informational abstraction, thus transforming the input messages into shorter ones; an example of this is provided by the experiment on the two-level switch architecture in Section 6. Consider for example an architecture in which a basic behavioral module receives from its sensors four-bit messages saying where the light is. If this basic behavioral module sends to the upper level four-bit messages indicating the proposed direction of motion, then the upper level could have used the sensorial information directly, by-passing the basic module. In fact, even if this basic behavioral module learns the correct input-output mapping, it does not operate any information abstraction and, as it sends to the upper level the same number of bits it receives from its sensors, it makes the hierarchy computationally useless.

Shaping policies

The use of a distributed system, either flat or hierarchical, brings in the new problem of deciding a *shaping policy*, that is the order in which the various tasks are to be learned. There are two extreme choices:

- *holistic shaping*: the whole network of CSs is treated as a single system, with all components being trained together;
- *modular shaping*: each component is trained separately.

Intermediate choices are possible.

In principle, training different CSs separately makes learning easier; however, the shaping policy must be designed in a sensible way. Hierarchical architectures are particularly sensitive to the shaping policy; indeed, it seems reasonable that the coordination modules be shaped after the lower modules have learnt to produce the simple behaviors. The "two-level and three-level switch architecture" experiments in Section 6 show that in fact good results are obtained by: shaping the lower CSs, then "freezing" them and shaping the coordinators, and finally letting all components free to go on learning together.

6. Experiments in simulated worlds

In this and in the next section we present some results obtained with simulated and real agents. The desire to give an answer to the following questions has guided the choice of which experiments to discuss:

- *Architecture*: does decomposition in subtasks help the learning process?
- *Shaping policy*: how must shaping be structured? Can basic behaviors and coordination of behaviors be learned at the same time, or is it better to split the learning process into several distinct phases?
- *Architecture/Shaping*: is there any relation between the agent's architecture and the shaping policy to be used?
- *Architecture/Learning*: can an inappropriate architecture impede learning?
- *Architecture's scalability:* can the different architectural approaches we used in the first experiments be composed themselves to build more complex hierarchical structures?
- *Memory*: how can the agent solve problems that require it to remember what happened in the past?
- *Simulation/Real world*: are there major differences between the real and simulated worlds?

Some other important questions, like the learning of basic behaviors, were discussed in a previous paper (Dorigo & Schnepf, 1993).

This section is organized as follows. First, we explain our experimental methodology. Second, we illustrate the simulated environments we used to carry out our experiments. Third, we report experiments that try to answer the first four questions (about architecture, shaping and learning). Fourth we show the result of a first experiment about the scalability of our approach: a two-level switch architecture whose basic behavioral modules are monolithic architectures with distributed input. Fifth, we illustrate the results of the "find hidden object" experiment in the simulated world. This experiment has also been run with the physical robot (see Section 7). Last, we report some experiments about memory management. Real world experiments will be discussed in the next section.

Experimental methodology

Experiments in the simulated worlds were run at least until there was some evidence that the performance was unlikely to improve further; this evidence was collected automatically by a steady-state-monitor routine, checking whether in the last k cycles the performance had significantly changed. In experiments involving multi-phase shaping strategies, a new phase was started when the steady-state-monitor routine signaled that learning had reached a steady state. In the real world, experiments were run until either the goal was achieved or the experimenter was convinced that the robot was not going to achieve the goal (at least in a reasonable time). Simulation experiments were repeated several times (typically five), and we report the graphs of typical results. In fact, the use of the steady-state-monitor routine made it difficult to show averaged graphs, as new phases started at different cycles in different experiments. Nevertheless, all the graphs obtained were very similar, which makes us confident that the typical result we present is a good approximation of the average behavior of our learning system. Experiments with the real robots were repeated only occasionally, as they are highly time consuming. Also in this case, the experiments which were repeated showed that the differences between different runs were marginal.

In all the experiments in simulated worlds, we used $P = \frac{\text{Number of correct responses}}{\text{Total number of responses}} \leq 1$ as performance measure. That is, performance was measured as the ratio of correct moves to total moves performed from the beginning of the simulation. Note that the notion of "correct" response is implicit in the RP: a response is correct if and only if it receives a positive reward. Therefore, we call the above defined ratio the *cumulative performance measure induced by the RP*.

We usually plot the performance of the basic behaviors, of the coordination behaviors (when applicable) and of the global system. For basic and coordination behaviors only the moves in which they were active are considered; instead, the global performance is computed as the ratio of globally correct moves to total moves from the beginning of the simulation, where at every cycle (we call cycle the interval between two sensors readings) a globally correct move is a move correct with respect to the current goal. So, for example, if after ten cycles the Chase behavior has been active for 6 cycles proposing a correct move 4 times, and the Escape behavior has been active for 4 cycles proposing a correct move 3 times, then the Chase behavior performance is 4/6, the Escape behavior performance 3/4, and the global performance (4+3)/(6+4) = 7/10.

Simulation environments

From Section 5 it is clear that, in order to test all the proposed architectures, we need many different simulated worlds. As we need a basic task for each basic behavior, in designing the experimental environments we were guided, besides the desire of investigating pieces of the

Animat problem, by the necessity of building environments in which basic tasks, and their coordination, could be learned by the tested agent architecture. We used the following environments.

- Chase_an_object environment (single behavior environment, with and without memory).
- Chameleon/Chase environment (two behaviors environment).
- Chase/Feed/Escape environment (three behaviors environment).
- Find_hidden_object environment (two behaviors environment).

In the Chase_an_object environment (see Figure 11a), the task is to learn to follow a moving object. This environment was studied primarily to test the learning classifier system capabilities and as a test-bed to propose improvements in the CS model. These aspects and results have been presented and discussed for example in Dorigo & Sirtori (1991) and in Dorigo (1993, 1992c). Here it is sufficient to say that the analysis of this and related tasks led to the introduction of some new operators that significantly improved the learning performance, and that the resulting system was powerful enough to allow the real-time learning of simple behaviors like the light approaching one (see the experiments with the AutonoMouse in Section 7). Moreover, we have used this environment to test whether the addition of sensor memory could improve the performance of the agent. As we show at the end of this section, the results are promising.

The Chameleon/Chase environment was introduced to study the composition of two independent behaviors. In this environment the agent learns to follow a light source and to change its color according to the background color (see Figure 11b). Results obtained in this environment were very satisfying (see Dorigo, 1992c); using the flat architecture (Figure 8a) the agent was able to learn to follow the light source and to change its color correctly. (After 80,000 iterations, the average performance in the last 1,000 iterations was: 0.97 for the Chase behavior, 0.95 for the Chameleon behavior, and 0.92 for the Global system. These results may give the impression the learning algorithm is rather slow. On the contrary, a very good performance, higher than 0.8, was obtained after 7,000 iterations.) Details about these experiments will not be discussed in this paper because of space constraints.

In the Chase/Feed/Escape environment, already partially introduced in Section 3, there are three objects: a light, a food source and a predator. Basically, the robot is predisposed to follow the moving light source. When its distance from the food source is less than a threshold, then the robot feels hungry and thus focuses on feeding. When a predator appears, then the main goal is to run away from the predator. The maximum speed of the robot is the same as the speed of the light source and of the predator. The light source and the food are always present (but the food can be seen only when closer than a threshold). The predator appears at random time intervals, remains in the environment for a random number of cycles, and then disappears. The environment dimensions are 640 x 480 pixel and the food distance threshold was set to 45 pixels (a robot's step was set to 3 pixel). In Figure 11c a snapshot of the environment is shown.

In the Find_hidden_object environment the agent's goal, as in the Chase_an_object environment, is to follow a moving light source. The task is complicated by the presence of a wall. Whenever it is interposed between the light and the agent (see Figure 11d), the agent cannot see the light any longer, and must activate a new behavioral pattern, namely a Search_for_object behavior.

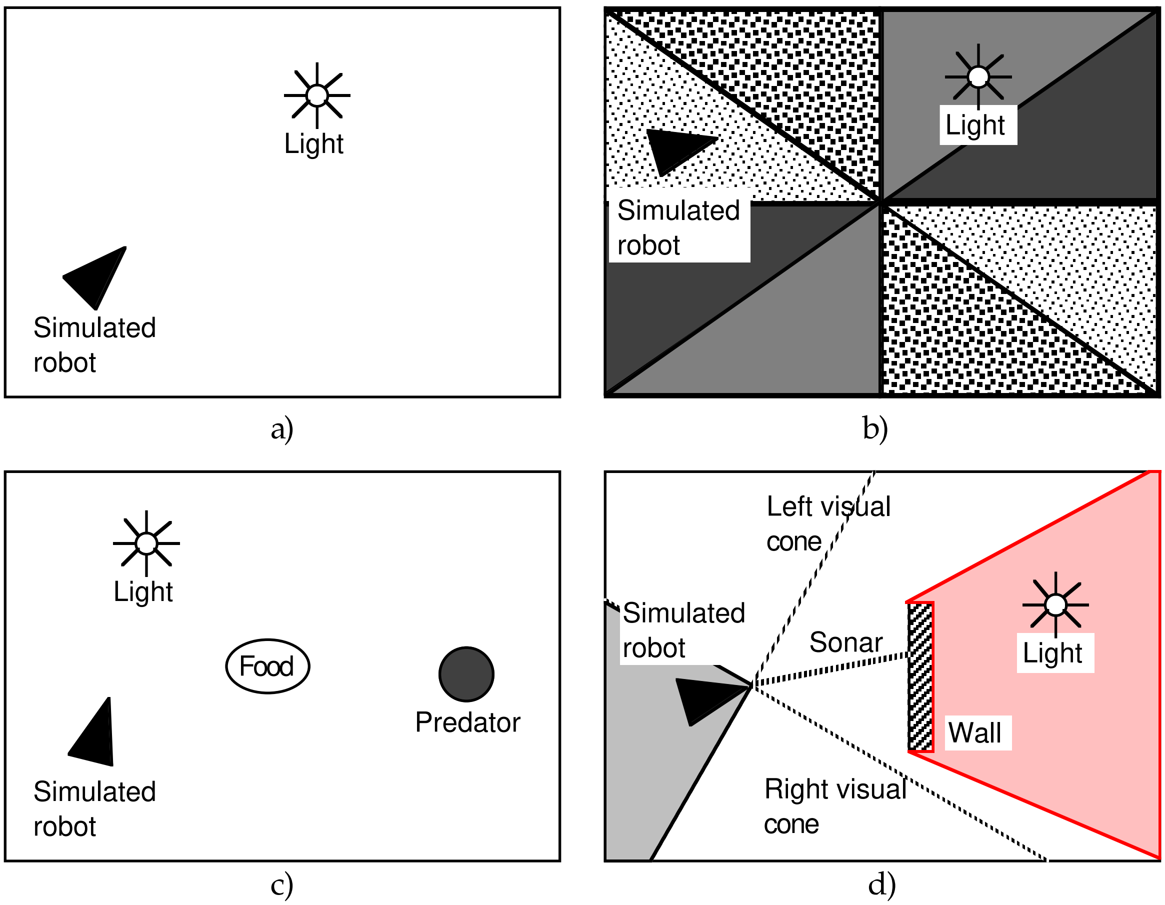


Figure 11. *Simulated environment setup:*
a) Chase_an_object environment.
b) Chameleon/Chase environment; the environment was partitioned into eight sectors of four different colors.
c) Chase/Feed/Escape environment.
d) Find_hidden_object environment: the agent does not see the light when it is in the shaded area.

The issues of architecture, shaping and learning

Our experiments show that a factorization of the learning task into several simpler learning tasks helps. This is obvious, though it is still interesting to see to what extent cooperation among the modules comprising the learning system can itself be learned. As discussed in Section 5, two architectural decisions must be taken by the system designer: how to decompose the learning problem, and how to make the resulting modules interact. The first issue is a matter of efficiency: a basic behavior should not be too difficult to learn. In our system, this means that classifiers should be no longer than about 30 bits (and therefore messages cannot be longer than 10 bits). The second issue is both a matter of efficiency, comprehensibility, and learnability. We feel, though this was not proved experimentally because we did not reach the complexity required by such an experiment, that a coordination module is constrained by the same limitations in complexity as basic modules. The longer the message[3] received, the longer the time required to learn. Comprehensibility means that by examination of the architecture a human observer should be able to understand why certain connections occur. Learnability refers to the fact, already discussed in Section 5, that not every architecture allows the system to learn any behavior.

Monolithic architecture

The monolithic architecture is the most straightforward way to apply CSs to our learning problem; just have a single CS learn the whole thing. With this approach the machinery provided by ALECSYS is redundant. Results obtained with the monolithic architecture will therefore be used as a reference to evaluate whether by decomposing the overall task into simpler subtasks, and/or by using a hierarchical architecture, we obtain improved performance. In an attempt to be fair in comparing the different approaches, we adopted the same number of transputers in every experiment[4].

Figure 12 shows the typical result for the Chase/Feed/Escape environment. An important observation is that the performance of the Escape behavior is higher than the performance of the Chase behavior, which in turn is higher than that of the Feed behavior. This result holds for all the experiments with all the architectures. The reason is twofold. The Escape behavior is easier to learn because our agent must learn to choose the escaping movement among 5 out of 8 possible directions, while the correct directions to Chase an object are, for our agent, 3 out of 8 (see Figure 13).

The lower performance of the Feed behavior is explained by the fact that, in our experiments, the agent could see the object to be chased and the predator from any distance, while the food

[3] The length of a message received by a coordination module is proportional to the number of lower level modules coordinated and to the quantity of information each lower level sends to it.

[4] For a given number of processors, the system performance is dependent on the way the physical processors are interconnected, i.e. on the hardware architecture. The hardware architecture we use was chosen after a careful experimental investigation presented elsewhere (see Piroddi & Rusconi , 1992, and Camilli & others, 1990).

could be seen only when closer than a given threshold. This caused a much lower frequency of activation of Feed, that resulted in a slower learning rate for that behavior.

Another observation is that, after an initial very quick improvement of performance, both basic and global performances set to an approximately constant value, far from optimality. The global performance after 80,000 cycles reaches the value 0.72 and does not change any more (we ran the experiment up to 300,000 cycles without observing any improvement). In fact, as classifiers are 51 bits long, the search space, i.e., the cardinality of the set of possible different classifiers, in this architecture has dimension $3^{34} * 2^{17}$. The genetic algorithm, together with the apportionment of credit system, appears unable to search such a huge space in a reasonable time.

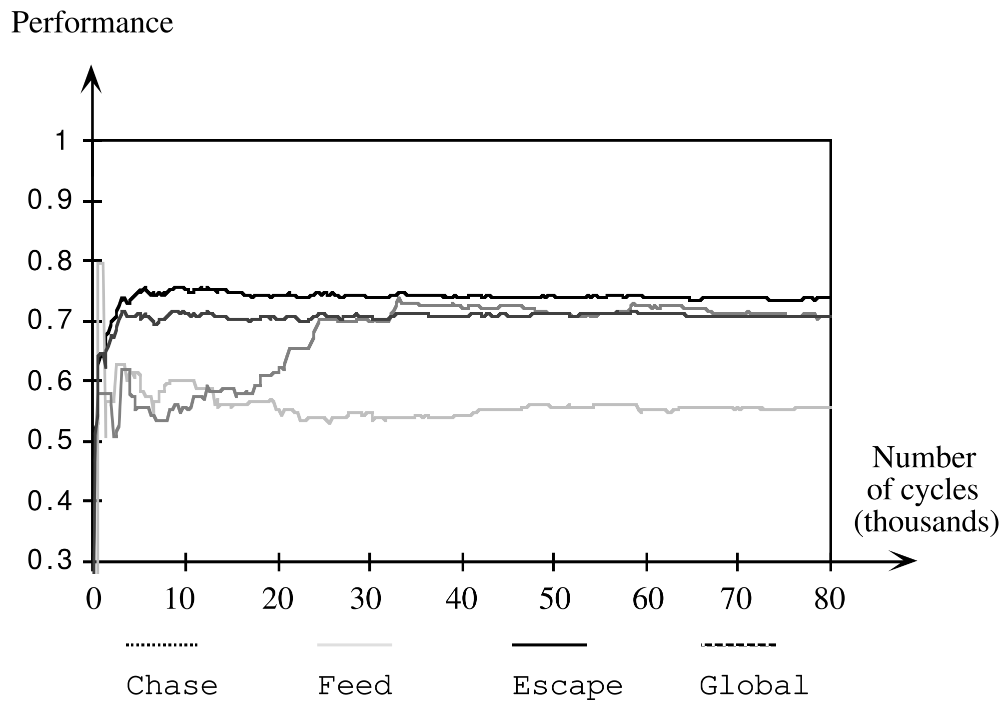


Figure 12. *Cumulative performance of the typical experiment with the monolithic architecture.*

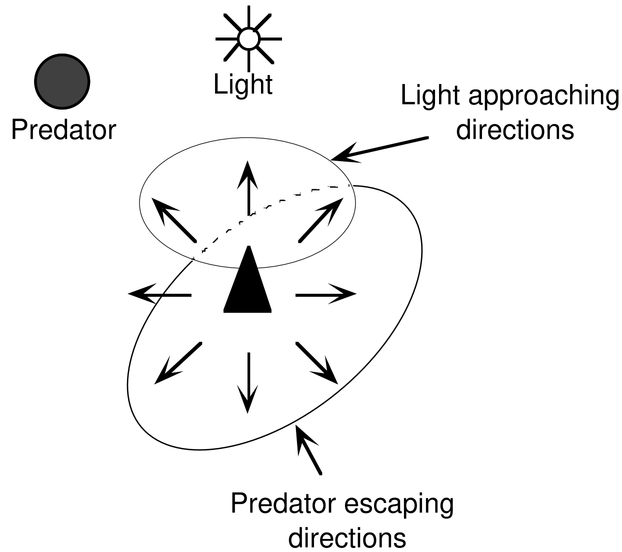


Figure 13. *Difference between approaching and escaping behaviors.*

Monolithic architecture with distributed input

With this architecture environmental messages are shorter (5 bits long) than in the previous case, and we expect therefore a better performance. More than one message can be appended to the message list at each cycle (maximum three messages, one for each basic behavior).

The results, shown in Figure 14, confirmed our expectations: global performance settled to 0.86 after 80,000 cycles and both the Chase and Escape behaviors reached higher performance levels than with the previous monolithic architecture. Only the Feed behavior did not improve its performance. This was partially due to the early stop of the experiment. In fact, in longer experiments, in which it could be tested adequately, the Feed behavior reached a higher level of performance, comparable with that of the Chase behavior. It is also interesting to note that the graph qualitatively differs from that of Figure 12; after the initial steep increase, performance continues slowly to improve, a sign that the learning algorithms are effectively searching the classifiers space.

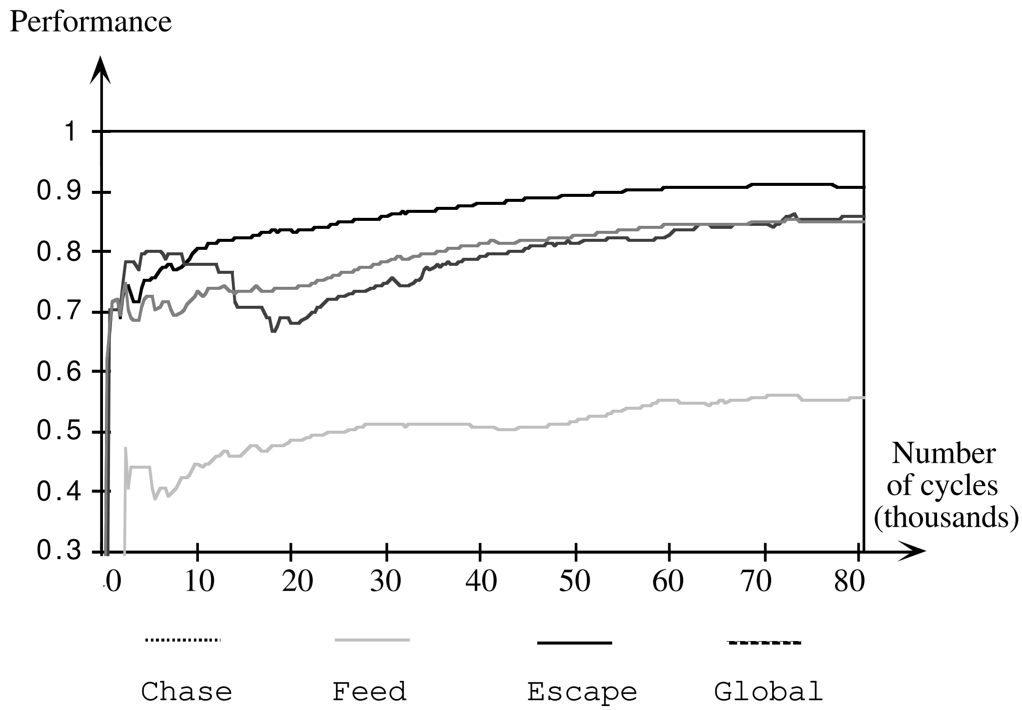


Figure 14. *Cumulative performance of the typical experiment with the monolithic architecture with distributed input.*

Two-level switch architecture

In this experiment we used a two-level switch architecture, in which the coordination behavior implemented suppression. The results, reported in Figures 15 and 16, give the following interesting information. First, as shown in Figure 15 where we report the performance of the three basic behaviors and of the coordinator (switch) in the first 50,000 cycles and the global performance from cycle 50,000 to the end of the experiment, the use of the holistic shaping policy results in a final performance that is comparable to that obtained with the monolithic architecture. This is due to the fact that with holistic shaping rewards obtained by each individual CS are very noisy. In fact, with this shaping policy we give each CS composing the agent the same reward, computed observing the global behavior. This means that there are occasions in which a double mistake results in a correct, and therefore rewarded, final action. Consider for example the situation in which Escape is active and proposes a (wrong) move towards the predator, but the coordinator fails to choose the Escape module and chooses instead the Chase module, which in turn proposes a move away from the chased object (wrong move), say in the direction opposite to that of the predator. The result is a correct move (away from the predator) obtained by the composition of a wrong selection of the coordinator with a wrong proposed move of two basic behaviors. It is easy to understand that it is difficult to learn good strategies with such a reward function.

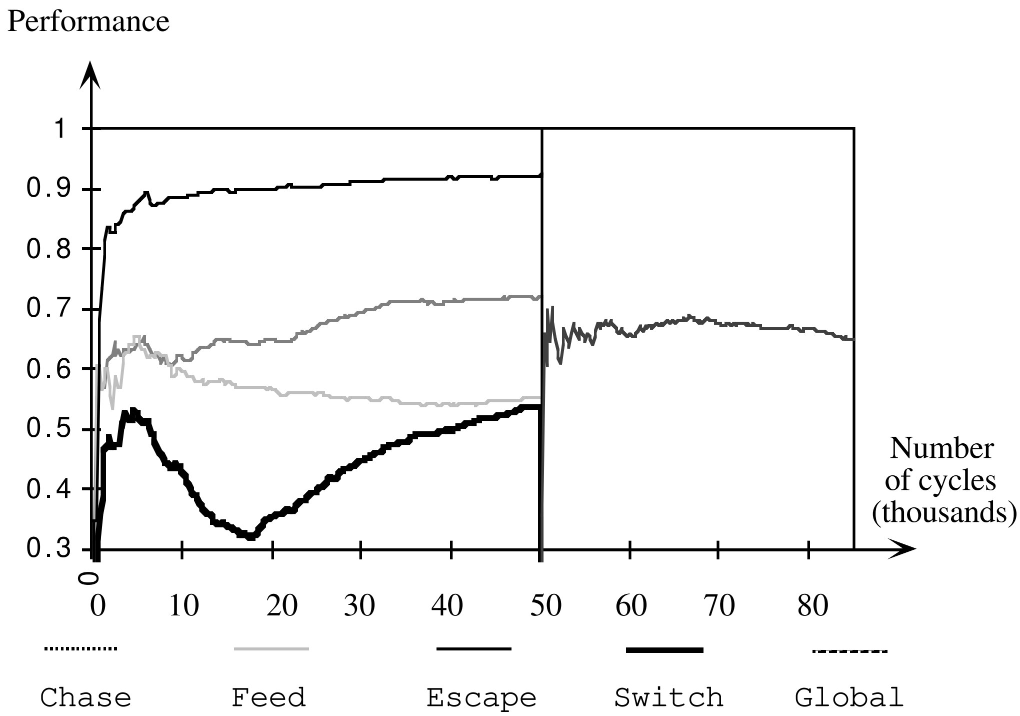


Figure 15. *Cumulative performance of the typical experiment with the two-level switch architecture. Holistic shaping.*

Second, using the modular shaping policy, performance improves, as expected. The graph of Figure 16 shows three different phases. In the first one, during the first 33,000 cycles, the three basic behaviors were independently learned. Between cycles 33,000 and 48,000 they were "frozen", i.e., learning algorithms were deactivated, and only the coordinator was learning. After cycle 48,000 all the components are free to learn, and we observe the global performance. The maximum global performance value obtained with this architecture was 0.84.

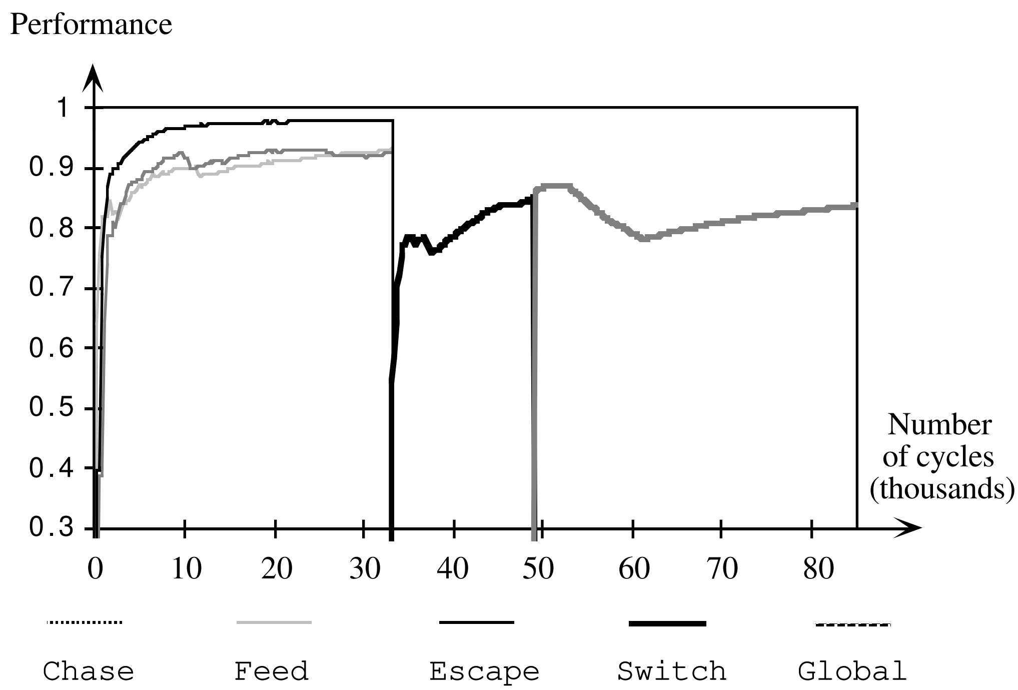


Figure 16. *Cumulative performance of the typical experiment with the two-level switch architecture. Modular shaping.*

As a help to the reader, we summarize in Table 1 the results about monolithic and two-level architectures already presented in Figures 12, 14, 15 and 16. A problem in filling this table was that the experiments were not run using the same number of iterations. This, as already said, is due to the steady-state-monitor routine, which automatically decided when to shift to a new phase of the experiment or when to stop it. Still, the first four experiments are comparable, as they all run for about 80,000 iterations. Table 1 shows that, from the global behavior point of view, the best results were obtained by the *monolithic architecture with distributed input* and by the *two-level switch architecture with modular shaping*. The performance of the basic behaviors in the second was always better. In particular the Feed behavior achieved a much higher performance level; in fact, using the two-level switch architecture with modular shaping, each basic behavior is fully tested independently of the others, and therefore the Feed behavior has enough time to learn its task. It is also interesting to note that the *monolithic architecture* and the *two-level switch architecture with holistic shaping* have roughly the same performance.

Table 1. *A comparison of monolithic and two-level hierarchical architectures. Performance is measured as the cumulative ratio of the number of correct moves to the total number of responses produced. As experiments were run with different total numbers of iterations, the number of iterations used to compute the performance is shown in parentheses under the performance value.*

Architecture	Chase	Feed	Escape	Switch 1	Global
Monolithic	0.71 (80000)	0.56 (80000)	0.75 (80000)	- -	0.72 (80000)
Monolithic with distributed input	0.85 (80000)	0.56 (80000)	0.92 (80000)	- -	0.86 (80000)
Two-level switch Holistic shaping	0.73 (50000)	0.56 (50000)	0.93 (50000)	0.54 (50000)	0.66 (85000)
Two-level switch Modular shaping	0.93 (33000)	0.93 (33000)	0.98 (33000)	0.84 (15000)	0.84 (85000)

Three-level switch architecture

The three-level switch architecture (see Figure 10) stretches to the limit the hierarchical approach (a three behaviors task architecture with more than three levels seems in fact to be senseless). Within this architecture the coordinator used in the previous architecture was split into two simpler, binary, coordinators. Using holistic shaping, results have shown that two- or three-level architectures are equivalent (see Figures 15 and 17, and Table 2). More interesting are the results obtained with modular shaping. As we have three levels, we can organize modular shaping in two or three phases. With two phases modular shaping basically we follow the same procedure used with the two-level hierarchical architecture; in the second phase basic behavioral modules are frozen and the two coordinators learn at the same time. In three phases modular shaping the second phase is devoted to shape the second level coordinator (all the other modules are frozen), while in the third phase the third level coordinator alone learns. Somewhat surprisingly, the results show that, given the same amount of resources (computation time in seconds), two phases modular shaping gave slightly better results. The reason stems from the fact that, while with two phases modular shaping both coordination behaviors are learning for the whole learning interval, with three phases modular shaping the learning interval is split into two parts during which only one of the two coordinators is learning, and therefore the two switches cannot adapt to each other. Graph of Figure 18 shows the very high performance level obtained in this way.

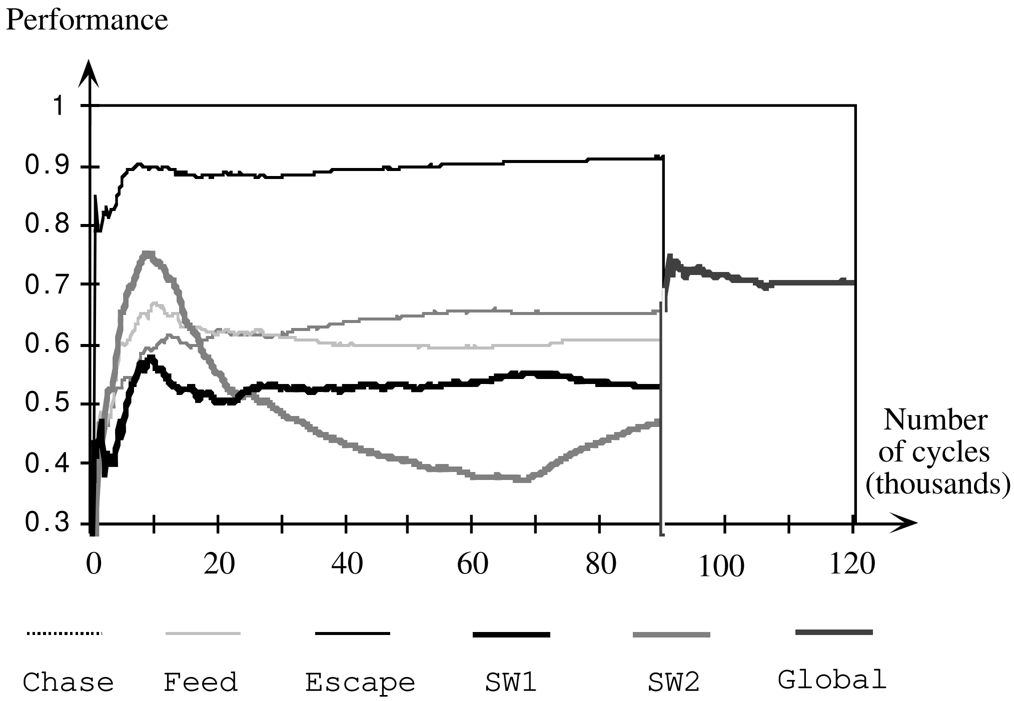


Figure 17. *Cumulative performance of the typical experiment with the three-level switch architecture. Holistic shaping using the architecture of Figure 10.*

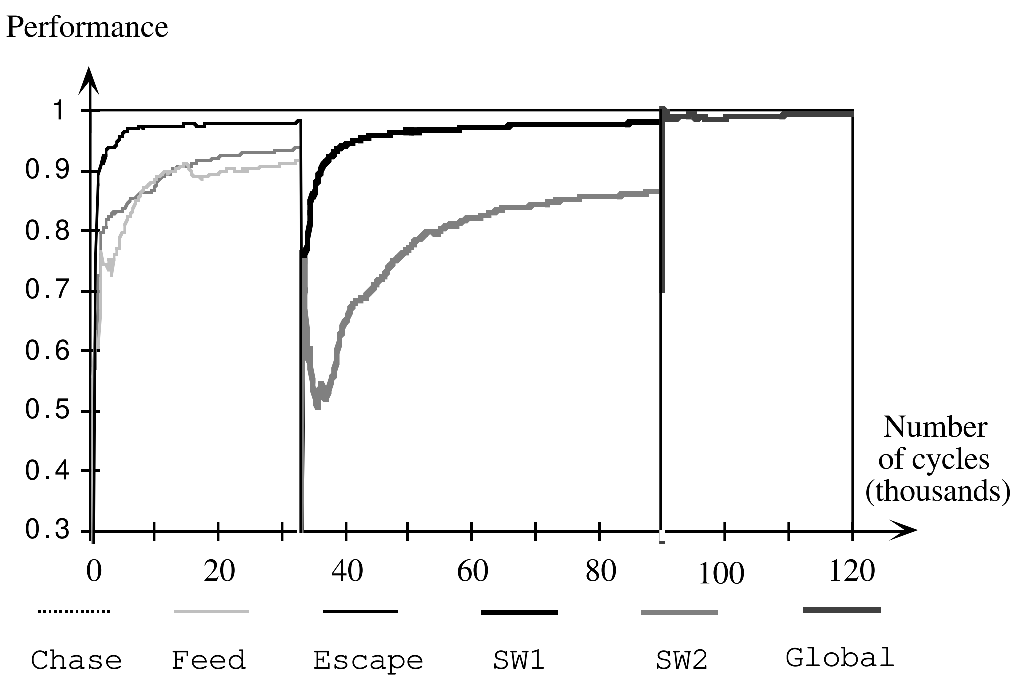


Figure 18. *Cumulative performance of the typical experiment with the three-level switch architecture. Two phases modular shaping using the architecture of Figure 10.*

For the reader's convenience, we compare in a table the results obtained with the two- and three-level switch architectures. Table 2 reports the performance of basic behaviors, of switches, and of the global behavior, as measured after k iterations, where k is the number in parentheses below each performance value.

Table 2. *A comparison of two- and three-level hierarchical architectures. Performance is measured as the cumulative ratio of the number of correct moves to the total number of responses produced. As experiments were run with different total numbers of iterations, the number of iterations used to compute the performance is shown in parentheses under the performance value.*

Architecture	Chase	Feed	Escape	Switch 1	Switch 2	Global
Two-level switch Holistic shaping	0.73 (50000)	0.56 (50000)	0.93 (50000)	0.54 (50000)	-	0.66 (85000)
Two-level switch Modular shaping	0.93 (33000)	0.93 (33000)	0.98 (33000)	0.84 (15000)	-	0.84 (85000)
Three-level switch Holistic shaping	0.66 (90000)	0.61 (90000)	0.92 (90000)	0.53 (90000)	0.48 (90000)	0.70 (120000)
Three-level switch Two phases modular shaping	0.94 (33000)	0.92 (33000)	0.98 (33000)	0.97 (56000)	0.86 (56000)	0.99 (120000)
Three-level switch Three phases modular shaping	0.93 (33000)	0.93 (33000)	0.98 (33000)	0.80 (10000)	0.80 (10000)	0.95 (120000)

We have run an experiment with the Chase/Feed/Escape behavior using the three-level switch architecture of Figure 19 to show that the choice of an agent architecture which does not correspond naturally to the structure of the target behavior leads to poor performance. This architecture differs from the architecture of Figure 10 because it was designed so that the distribution of tasks between SW1 and SW2 should impede learning. In fact, as SW2 does not know whether SW1 is proposing a Chase or an Escape action, it cannot decide (and therefore learn) whether to suppress SW1 or the Feed behavioral module.

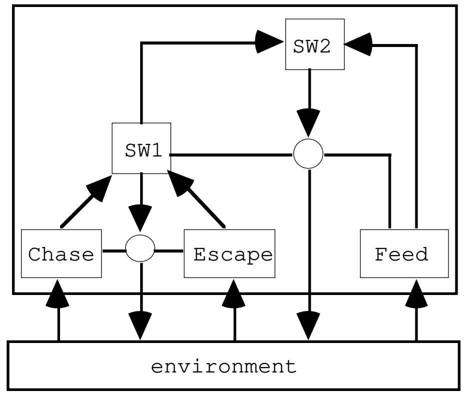


Figure 19. *A three-level switch architecture with a wrong disposition of coordination modules.*

Results are shown in Figures 20 and 21. As in the preceding experiment, two phase shaping gave better results than three phase. It is clear from Figure 21 that the low level of global performance achieved was due to the impossibility for SW2 to learn to coordinate the SW1 and the Feed modules.

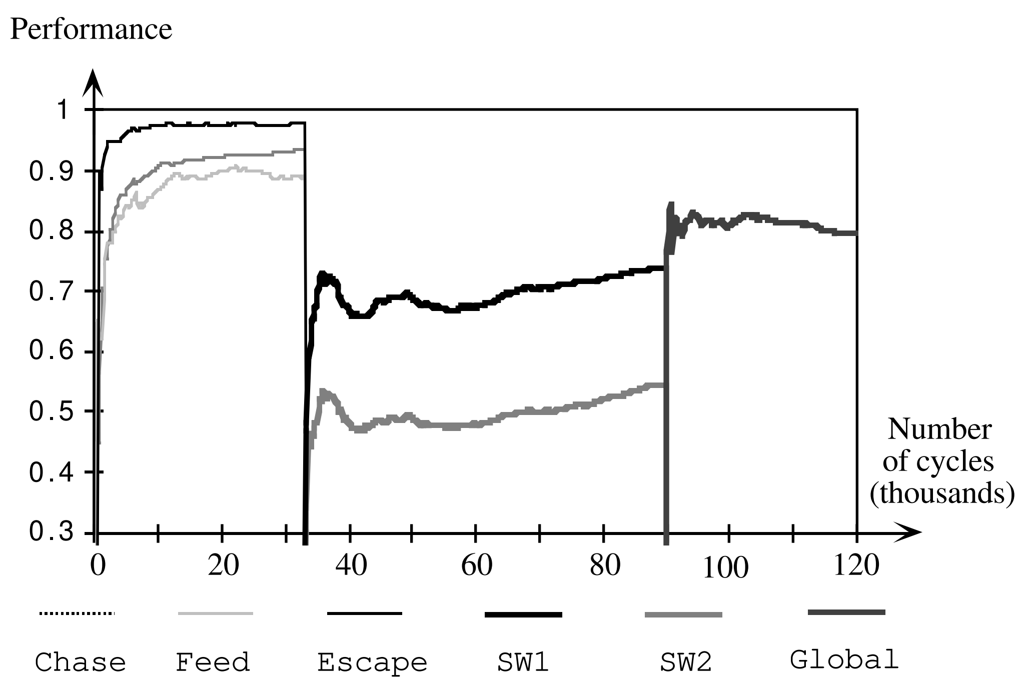


Figure 20. *Cumulative performance of the typical experiment with the three-level switch architecture. Two phases modular shaping using the "unnatural" architecture of Figure 19.*

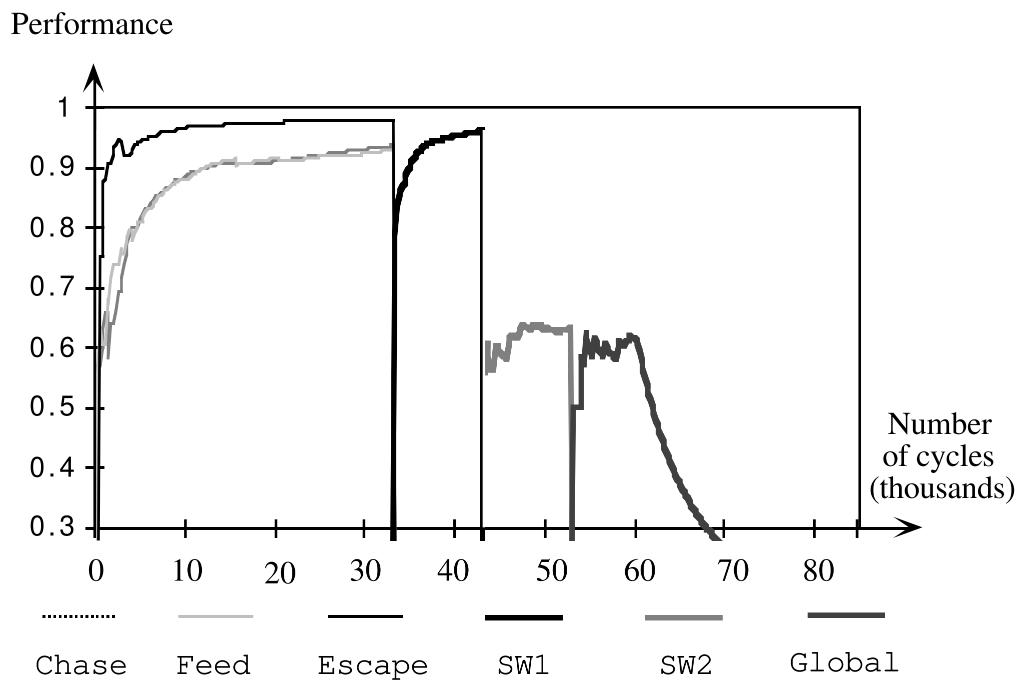


Figure 21. *Cumulative performance of the typical experiment with the three-level switch architecture. Three phases modular shaping using the "unnatural" architecture of Figure 19.*

The issue of scalability

The experiment presented in this section regards the composition of the monolithic architecture with multiple inputs with the two level hierarchical architecture. We used a Chase/Feed/Escape environment in which there were four instances of each class of objects (lights, food, predators). Only one instance in each class was relevant for the learning agent (i.e., the agent likes only one of the four light colors and of the four kinds of food, and fears only one of the four potential predators). Therefore, basic behaviors, in addition to the basic behavioral pattern, had to learn also to discriminate between different objects of the same class. For example, the Escape behavior, instead of receiving a single message indicating the position of the predator (when present), now receives messages regarding many different "animals", only one of which is a real predator. Different "animals" are distinguished by a tag, and Escape must learn to run away only by the real predator (it should be unresponsive to other animals). The experiments have shown (see Figure 22) that the agent learns the new, more complex, task, although the performance level is slightly lower than in the previous experiment of Figure 16.

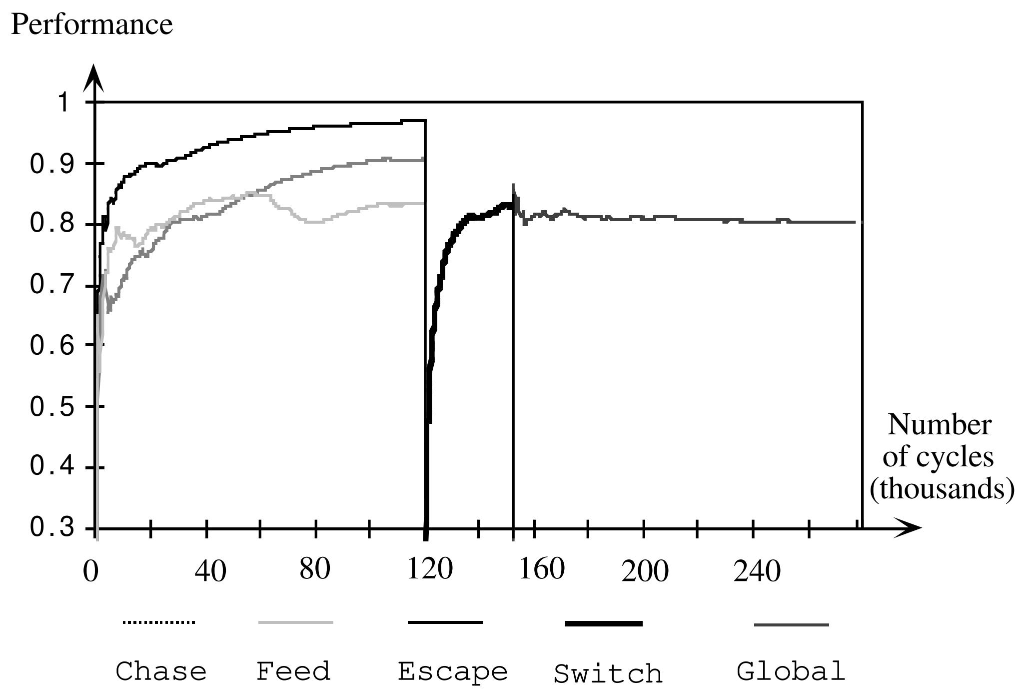


Figure 22. *Cumulative performance of the typical experiment with a multi input, two-level switch architecture. Modular shaping.*

Finding a hidden object

This experiment, whose environment is sketched in Figure 11d, has been run in two different versions, one by simulation and one with a real robot (AutonoMouse IV, see Figures 2 and 3b). The aim was to see whether our system was capable of learning a reasonably complex task, involving obstacle detection by sonar and whiskers and searching for a hidden object. The target behavior was to approach a light, walking around a wall when necessary. In these experiments we paid no attention to the issue of architecture, and adopted a simple monolithic CS throughout.

In both the simulated and the real experiment, the eyes and the sonar of the robot (AutonoMouse IV in the real world) where used as on/off sensors; the input interface included:

- 1 bit for each of the two eyes (used as on/off light sensors); each eye had a visual cone of 180 degrees, with a 90 degrees overlapping in front;
- 1 bit for the sonar;
- 1 bit for each of the two side whiskers.

The output interface included 2 bits, coding the following four possible moves: still, straight ahead, ahead with a left turn, and ahead with a right turn. In the simulated experiments, the wall had a fixed position, while the light automatically hid itself behind the wall each time it was reached by the agent. To shape the agent, the reinforcement program was written with the following strategy in mind:

```
if a light is seen
  then {Approach_the_light behavior}
      Approach it
  else {Search_for_object behavior}
      if a distal obstacle is sensed (by sonar)
        then
          Approach it
        else
          if a proximal obstacle is sensed (by whiskers)
            then
              Move along it
            else
              Turn consistently (go on turning in the same direction, whichever it is).
```

The distances of the robot from the light and from the wall were computed from the geometric coordinates of the simulated objects. The simulation was run for 50,000 cycles. In Figure 23 we separately show the Approach_the_light performance (when the light is visible), the Search_for_object performance (when the light is not visible) and the global performance. Approaching the light appears to be easier to learn than searching for it. This is easy to explain given that searching for the light is a rather complex task, involving moving toward the wall, moving along it and turning around when no obstacle is sensed.

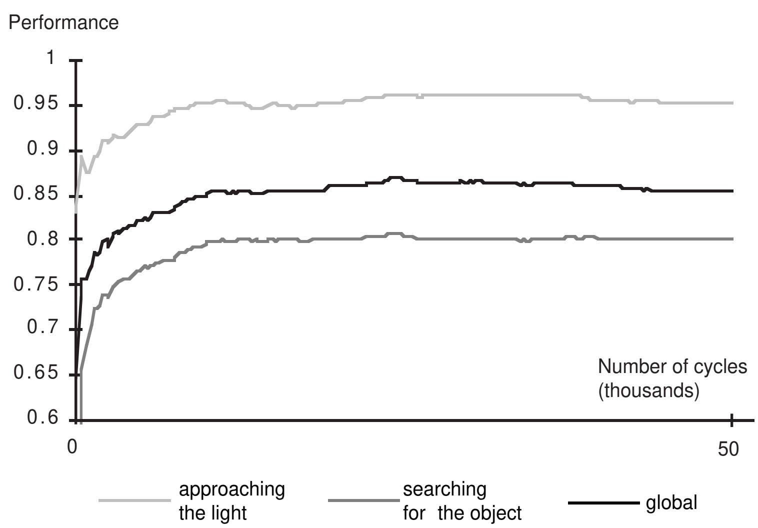


Figure 23. *Cumulative performance for the "finding a hidden object" task.*

Adding memory to ALECSYS

All the experiments described so far concern S-R behavior, i.e. direct associations of stimuli and responses. Clearly, the production of more complex behavior patterns crucially involves the ability to deal with dynamic behavior, that is with input-output associations that exploit some kind of internal state. We have only started moving in this direction, but a few experiments deserve reporting.

In a dynamic system, a major function of the internal state is memory. Indeed, the limit of S-R behavior is that it can relate a response only to the current state of the environment. It must be noticed that ALECSYS is not completely without memory; in fact, both the strengths of classifiers and the internal messages appended to the message list embody information about past events. However, it is easy to think of target behaviors that require a much more specific kind of memory.

In Section 4, we have already argued that following a light can be made easier by a memory of past perceptions. We have endowed the learning system with a *sensor memory*, that is a kind of short-term memory of the state of the agent's sensors. In order to avoid an *ad hoc* solution to our problem, we have adopted a sensor memory that functions uniformly for all sensors, independently of the task. The idea was to provide the agent with a representation of the previous state of each sensor, for a fixed period of time. That is, at any given time t the agent can establish, for each sensor S, whether:

(i) the state of S has not changed during the last k cycles (where the *memory span* k is a parameter to be set by the experimenter);
(ii) the state of S has changed during the last k cycles; in this case, enough information is given so that the previous state of S can be reconstructed.

This design allows us to define a sensor memory that depends on the input interface, but is independent of the target behavior (with the exception of k, whose optimal value can in fact be a function of the task). More precisely, the sensor memory is made up of:

- a memory word, isomorphic to the input interface;
- an algorithm that updates the memory word at each cycle, in accordance to the specifications (i) and (ii), on the basis of the current input, of the previous memory word, and of the number of cycles elapsed from the last change;
- a mechanism that appends the memory word to the message list, with a specific tag identifying it as a memory message.

The memory process is described in more detail in Figure 24. Note that, coherently with our approach, the actual "meaning" of memory messages must be learnt by the classifier systems. In other words, memory messages are just one more kind of messages, whose correlation with the overall task has to be discovered by the learning system.

The results obtained in a typical simulation are reported in Figures 25–27, that compare the performances of the agent with and without memory. The target behavior was to track a moving light. The agent had two frontal eyes, each with a visual cone of 60 degrees, overlapping for 30 degrees; as a result, the visual space in front of the agent was 90 degrees, partitioned into three sectors of 30 degrees each. The memory span was set to 10. Figure 25 shows that the performance of the agent with memory tends to become asymptotically better than the one of the memoryless agent. However, the learning process is slower. This is easy to explain: the "intellectual task" of the agent with memory is harder, because the role of the memory messages has to be learned; on the other hand, the agent can learn about the role of memory only when it becomes relevant, that is when the light disappears from sight – and this is a relatively rare event. To show that the role of memory is actually relevant, in Figures 26 and 27 we have decomposed the agent's performance into: a) the performance produced when the light is not visible (and therefore memory is relevant); b) the performance when the light is visible (and thus memory is superfluous). It can be seen that in the former case the performance of the agent with memory is significantly better. We conclude that even a very simple memory system can improve the performance of ALECSYS in those cases in which the target behavior is not intrinsically S-R.

S	s1	···	sj	···	sN
M	m1	···	mj	···	mN

S = environment message, coding the state of the sensors; each sensor is represented by a bit sequence sj (N being the number of distinct sensors);
M = memory message; for each sj, there is an mj of equal length.

The memory message at time t, M(t), is computed from S(t), S(t-1), and M(t-1). The following algorithm is applied at each non initial cycle t (at cycle t=0, mj(0):=sj(0)):

```
for j from 1 to N do                    for each sensor j,
  delta_sj= sj(t) xor sj(t-1);            compute the change of sj from t-1 to t;
  if delta_sj ≠ [0 ... 0 ... 0]         if there is a change,
    then
      mj(t) := delta_sj;                  set mj to it
      clock(j) := 0                       and set the clock of sensor j to 0,
    else
      mj(t) := mj(t-1);                   else set mj to its previous value,
      clock(j) := clock(j) + 1            and increment the clock of sensor j.
  endif;
  if clock(j) > k                       if the memory span of sensor j has elapsed,
    then
      mj(t) := 0;                         set mj to zero
      clock(j) := 0                       and set the clock of sensor j to zero.
  endif
endfor
```

Figure 24. *The memory process.*

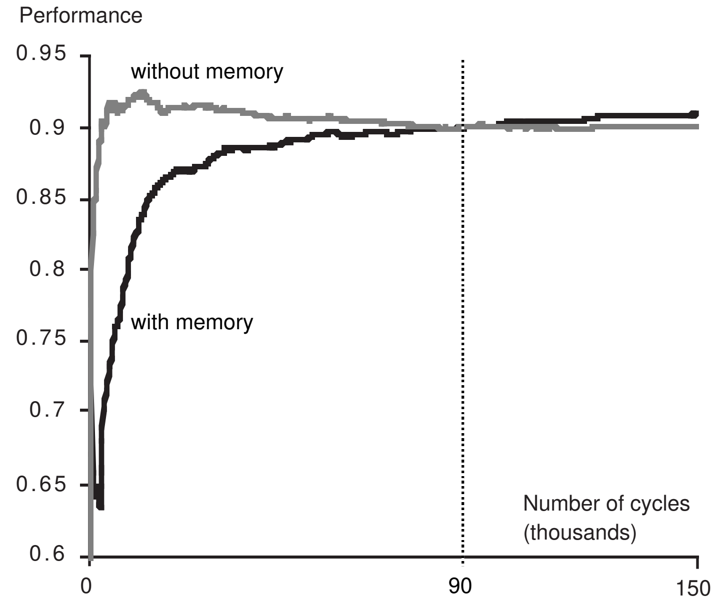


Figure 25. *Following a moving light with and without sensor memory.*

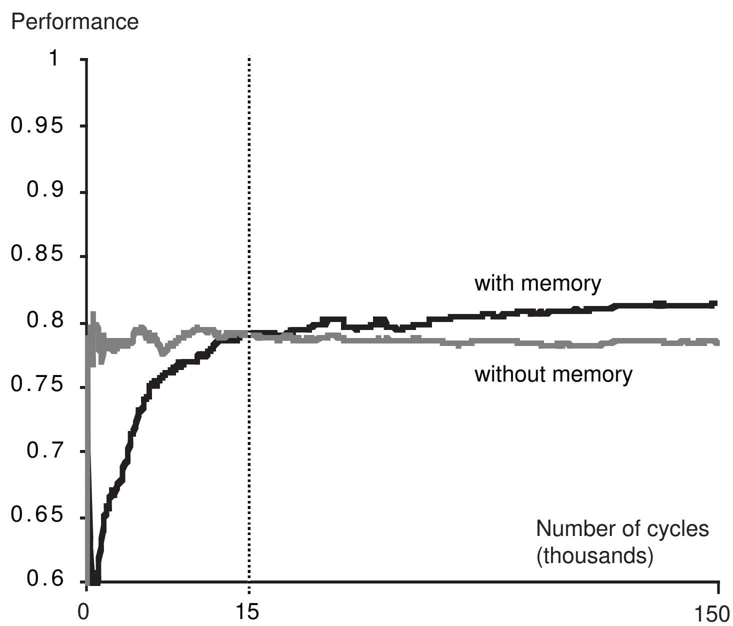


Figure 26. *Light following performance when the light is not visible.*

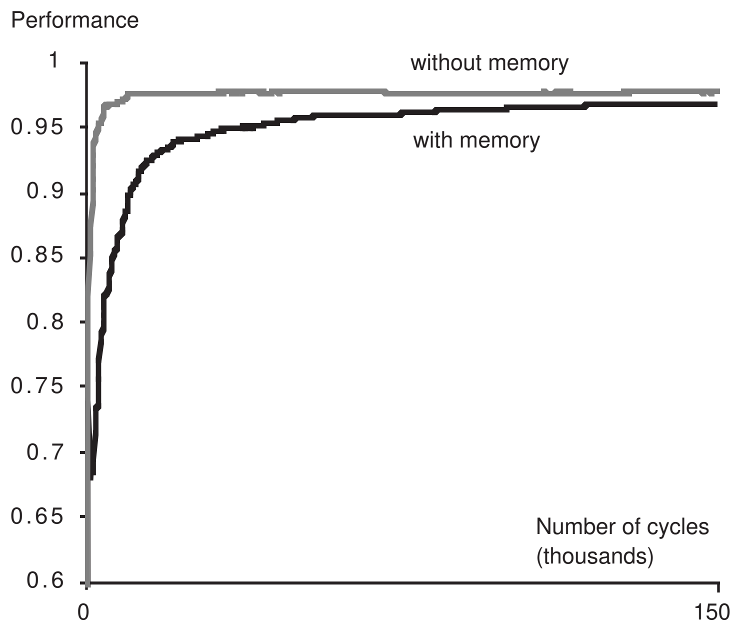


Figure 27. *Light following performance when the light is visible.*

7. Experiments with the real robot

Moving from simulated to real environments is challenging. Not only do the robot's sensors and actuators become noisy, but also the RP must rely on real, and hence noisy, sensors to evaluate the learning robot moves. We ran some experiments to see to what extent the real robot could use the ideas and the software used in simulations. In this section we present results from experiments in the real world using both AutonoMouse II and AutonoMouse IV.

Experiments with AutonoMouse II

As we said in Section 2, this version of the AutonoMouse is rather unsophisticated; essentially, it allows the design of very simple experiments, as the available sensors are only the four binary eyes and one binary ear.

Nevertheless, it was possible to show: (i) that our approach works also in real environments, where time constraints must be met and where sensor input and actuator output are affected by noise, and (ii) that our system is adaptive, being capable of graceful degradation of performance in presence of bad-working sensors or actuators. In all the experiments with AutonoMouse II we used a monolithic instantiation of ALECSYS, with characteristics (format of input and output messages, internal parameters used by the learning system, etc.) that are essentially the same as those used for the light approaching module of the experiments in the preceding section (see Figures 5 and 11a). In the experiments with AutonoMouse II, performance was measured through the "trainer's sensors" on board, that is by light intensity: when the robot approaches the light source, light intensity increases.

Approaching the light source

In the first experiment we position the AutonoMouse II in a room and let it move. The RP rewards the learning system whenever it approaches the light source, and punishes it in case of wrong moves. In these experiments with the real robot, the RP evaluates the approaching behavior using itself real sensors, i.e. the two central eyes of Figure 1. Graph of Figure 28 shows the developing approaching behavior. Performance is measured through light intensity (0 to 255). In the graph we also show the average reward (on the last 20 cycles) received by the learning system; in this experiment rewards are +50 for a correct move, –70 for a wrong one. The drop in performance at cycles 140 and 380 is due to a movement of the light source; as the AutonoMouse had reached the lamp, we moved it far away to continue the experiment. In this experiment 100 cycles took about 60 seconds. At cycle 220 the AutonoMouse started moving away from the light source because of some wrong classifiers; as the moves were wrong, ALECSYS was punished by the reinforcement program, and therefore the classifiers responsible for the wrong actions lost importance and finally were eliminated.

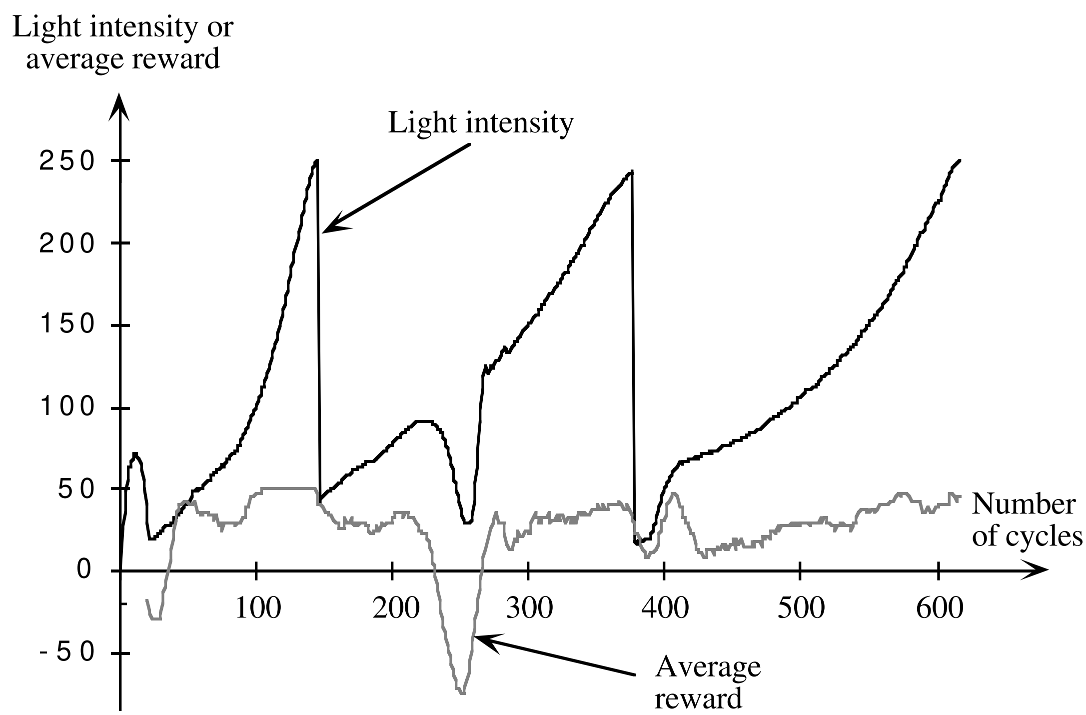


Figure 28. *The AutonoMouse II learns to approach a light source. Light intensity and average reward received.*

It is interesting that the number of cycles required to reach the light is significantly lower than the number of cycles required to reach a high performance in the simulation experiments. This is easily explained if you think that the correct behavior is more frequent than the wrong one as soon as performance is higher than 50%. The AutonoMouse starts therefore to approach the light source much before it has reached a high frequency of correct moves. Moreover, a comparison of the average reward graph with the light intensity graph reveals an interesting property of the real robot: its performance, measured as light intensity, shows some kind of inertia (with respect to average reward). In fact, it takes time to move and turn, and it is necessary to make many wrong moves to start to move away from the light source.

Approaching the light source with a blind eye

To test the adaptive capability we ran some experiments in which the AutonoMouse II's capabilities were degraded. These were:

- AutonoMouse with inverted eyes;
- AutonoMouse with inverted motors;
- AutonoMouse with one blind eye;

- AutonoMouse with non correct calibration of motors' speed (one motor is slower than the other one).

All of the experiments showed that the AutonoMouse, although with some degraded performance, was still capable of achieving the goal of approaching the light source. A thorough discussion of these experiments can be found in Dorigo (1992b, see also Colombetti & Dorigo, 1992). As an example we report here in Figure 29 the result of the one blind eye experiment. The number of cycles required to reach the light was slightly higher than before and at cycle 135 the AutonoMouse lost sight of the light, did a 360 degrees turn and started to approach the light again. Nevertheless, the AutonoMouse achieved its goal in a reasonably short time.

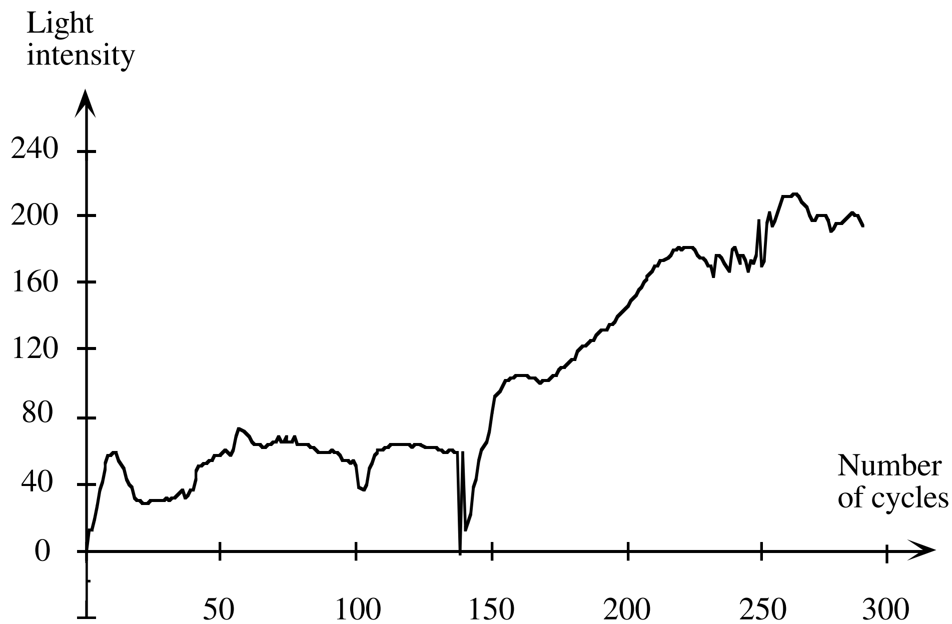


Figure 29. *A "one blind eye" AutonoMouse II learns to approach a light source.*

Experiment with AutonoMouse IV

With AutonoMouse IV we run the experiment on finding a hidden object, whose simulated counterpart is described in Section 6. The environment consisted of a large room containing an opaque wall, about 50 x 50 cm, and an ordinary lamp (50 W). The wall was realized as a pleated surface, in order to reflect back the sonar's beam coming from a wide range of directions (see Figure 30). The input and output interfaces were exactly the same as in the simulation, and so was the RP. The input from the sonar was defined in such a way that a front obstacle was detected within about 1.5 m from the robot.

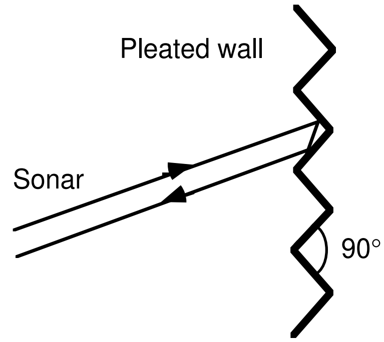


Figure 30. *Horizontal section of the pleated wall, and reflection of the sonar's beam.*

There were three main differences between the real and the simulated experiments. The first difference was that in the real environment the light was moved by hand by the experimenter, hiding it behind the wall when the AutonoMouse got very close to it (10–15 cm). In comparison with the simulated environment, where the light was moved by the simulation program in a systematic way, this procedure introduced an element of irregularity. From the results of the experiment, it is not easy to understand whether this irregularity affected the learning process.

Second difference: the distances of AutonoMouse IV from the light and from the wall were estimated on the basis of the outputs of the light sensors and of the sonar, respectively. More precisely, each of the two eyes and the sonar output an 8-bit number ranging from 0 to 255, respectively coding the light intensity and the distance from an obstacle. To estimate whether the robot got closer to the light, the total light intensity (that is, the sum of the outputs of both eyes) at cycle t was compared with the total light intensity at cycle $t-1$. The sonar's output was used in a similar way to estimate whether the robot got closer to the wall.

The eyes and the sonar were used in different ways by the agent and by the RP: from the point of view of the agent, all these sensors behaved as on/off devices; for the RP, the eyes and the sonar produced an output with higher discriminative power. Therefore, the same hardware devices were used as the trainer's sensors and, through a transformation of their outputs, as the agent sensors of the agent. The rationale of this choice has been explained in Section 4; we remark here that the main reason for providing the agent with a simplified binary input was to reduce the size of the learning system's search space, thus speeding up learning.

By exploiting the full binary output of the eyes and of the sonar, it was possible to estimate the actual effect of a movement toward the light or the wall. However, given the present sensory apparatus of AutonoMouse IV, we could not establish the real effect of a left or right turn; for these cases, the RP based its reward on the expected move, i.e. on the output message sent to the effectors, and not on the actual move. (A detailed discussion of the difference between rewarding the AutonoMouse according to estimated or real effects of actions can be found in Dorigo, 1992b.)

Finally, the third difference: due to practical reasons, the experiment with the real AutonoMouse was run for about 4 hours, covering only 5,000 cycles, while the simulated experiment was run for 50,000 cycles.

The graph of Figure 31 shows that the agent learned the target behavior reasonably well, as was in fact intuitively clear by direct observation during the experiment. There is however a main discrepancy between the results of the real and the simulated experiment. In the simulation, after 5,000 cycles the light approaching, light searching and global performances had respectively reached the approximate values of 0.92, 0.76 and 0.81; the three corresponding values in the real experiment are lower (about 0.75) and very close to each other. To put it differently, the real and the simulated light searching performances are very similar; on the contrary, while in the simulated experiment the light approaching behavior is much more effective than the light searching behavior, in the real experiment they are about the same.

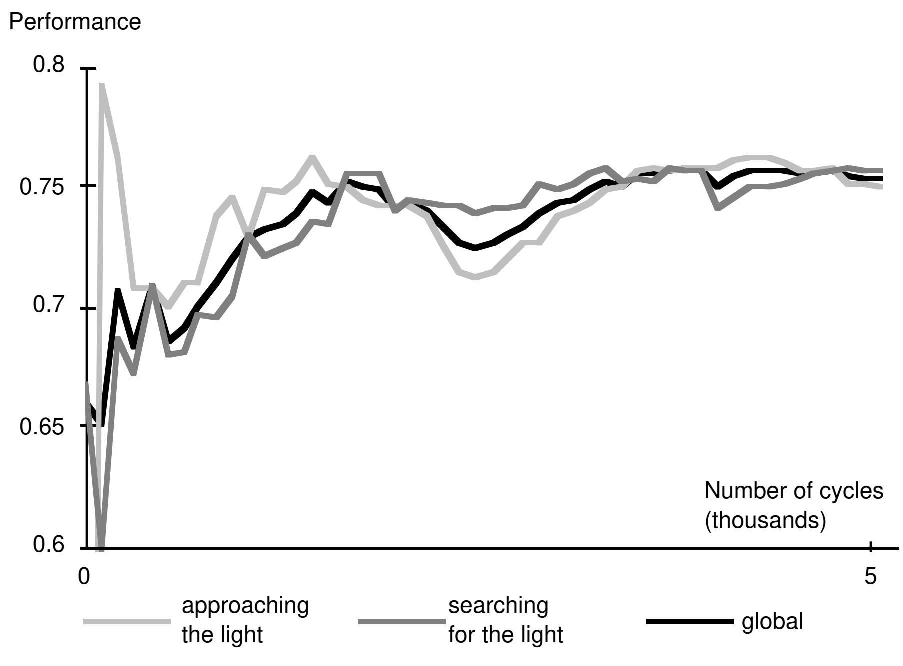


Figure 31. **Finding a hidden object by AutonoMouse IV.**

We interpret this discrepancy between the real and the simulated experiment as an effect of the different way in which the distance between the robot and the light was estimated. In fact, the total light intensity does not allow for a very accurate discrimination of such a distance. Often, a move toward the light did not result in an increase of total light intensity large enough to be detected; therefore, a correct move was not rewarded, because the RP did not understand that the robot did get closer to the light. As a consequence, the rewards given by the RP with respect to the light approaching behavior were not as consistent as in the simulated experiments. To check whether this hypothesis is correct, more experiments need to be run.

8. Comparison with related work

We have already pointed out the relationships between our work and research going on in the area of situated agents. In this section we relate our approach to other limitrophe research fields. Most prominent is the work on learning classifier systems (Booker, Goldberg & Holland, 1989; Booker, 1988; Wilson, 1987). We built on that work, introducing the idea of using a set of communicating classifier systems, running in parallel on a MIMD architecture. We also modified the basic learning algorithms to make them more efficient (a technical discussion of the learning algorithms can be found in Dorigo, 1993, 1992a, 1992b). More generally, the whole field of reinforcement learning is related to our work. Reinforcement learning has recently been studied in many different algorithmic frameworks, learning classifier systems being one. Notably, we have connectionist reinforcement learning (e.g., Barto, Sutton & Anderson, 1983; Williams, 1992), classifier systems reinforcement learning (e.g., Holland & Reitman, 1978; Robertson & Riolo, 1988; Booker, 1988; Dorigo, 1992b), and temporal differences reinforcement learning and related algorithms, like the adaptive critic heuristics (Sutton, 1984) and Q-learning (Watkins, 1989; Watkins & Dayan, 1992). These different approaches to reinforcement learning are often overlapping. For example, the adaptive critic heuristics and Q-learning have been implemented by means of a connectionist system in Lin (1992); also, Compiani & others (1989) have shown the existence of tight structural relations between classifier systems and neural networks.

Often the applications used to illustrate and compare the proposed algorithms are taken from the realm of autonomous robotics. A major difference with our work is that we do not investigate the temporal credit assignment problem, which is often a main point in reinforcement learning applications. Another difference is that only a few of the reinforcement learning applications deal with real robots. For example, Singh (1992), Lin (1992) and Millán & Torras (1992) use a point robot moving in a two dimensional simulated world; and Millán (this volume) uses a simulation of a *Nomad 200* robot. Grefenstette's SAMUEL, a learning system which uses genetic algorithms, (Grefenstette, Ramsey & Schultz, 1990), learns decision rules for a simulated task (a plane should learn to avoid being hit by a missile). Booker's GOFER system deals with a simulated robot living in a two-dimensional environment and whose goal is to learn to find food and avoid poison. The choice of using a real robot makes things very different, as the efficiency of the system becomes a major constraint. This constraint has guided our choice of a hierarchical architecture in which different modules can run in parallel (the same constraint has guided the subsumption architecture choice of Mahadevan & Connell, 1992).

Beside the mentioned work of Mahadevan & Connell, there are only a few other applications of reinforcement learning to real robots.

Maes & Brooks (1990) describe an algorithm to learn the coordination behavior of a six legged robot. Their algorithm is focused on learning coordination of hardwired basic behaviors, whereas in our case both basic behaviors and their coordination are learned. Brooks (1991b) has recently discussed the possibility to use genetic algorithms to evolve programs written in GEN, a high level language especially designed to produce programs which can be easily evolved by the genetic algorithm. (GEN can then be compiled into the Behavior Language (Brooks, 1990b), a rule-based parallel programming language which compiles into the subsumption architecture.) This idea is, to the authors' knowledge, still under development (Brooks, 1991b) and no results have been published yet.

An approach similar to that proposed by Brooks was taken by Koza & Rice, which used the Genetic Programming paradigm (Koza, 1992) to evolve Lisp programs to control an autonomous robot (Koza & Rice, 1992). Although they use genetic algorithms, their approach is very different from ours (and is much closer to the proposal of Brooks); in their case the genetic algorithm searches in the space of an opportunely defined subset of Lisp programs, while in our case the genetic algorithm is cast into the classifier system framework. They try to reproduce the results of Mahadevan & Connell (1992), applying their learning robot to the same problem. Nevertheless, their use of a simulated robot makes a fair comparison very difficult.

Also Beer & Gallagher (1992) have been using genetic algorithms to let a neural net learn to coordinate the movements of their six-legged robot. Also in this case the approach is rather different from ours, as they use the genetic algorithm to develop neural net controllers (see also Beer, this volume)

In his Ph.D. dissertation Kaelbling has been using reinforcement learning to let a robot – called Spanky – learn to approach a light source and to avoid obstacles (Kaelbling, 1990; but see also Kaelbling, 1991). She uses a statistical technique to store an estimate of the expected reinforcement for each action and input pair and some information of how precise that estimate is. Unfortunately, she reports only a qualitative description of the experiments in the real world. Her robots took from two to ten minutes to learn a good strategy, while AutonoMouse II after at most one minute was already pointing towards the light. Still, it is very difficult to make a comparison, as the experimental environment was not the same and Spanky's task was slightly more complex. The strength of our approach is that it allows for an incremental building of new capabilities; it is not clear whether this can be done with Kaelbling's approach.

Finally, the idea of shaping a robot is related to Shepanski and Macy's work (1987). They propose to train a neural net manually by interaction with a human expert. In their work the human expert substitutes our reinforcement program. This approach is very interesting, but it seems difficult to use in a nonsimulated environment; it is not clear therefore whether it can be adopted for real robot shaping.

9. Conclusions and future work

In this paper, we have described a possible approach to the development of situated agents through learning, and presented the results of experimental work aimed at demonstrating the viability of classifier systems and genetic algorithms for this purpose.

We view learning as a situated translation into a low-level control program of a higher-level conceptualization of a target behavior. Such a conceptualization is reflected into the reinforcement program, in charge of guiding the learning system through rewards and punishments. In our experiments, we have tried to enlighten some relationships holding among the target behavior, the agent and the trainer. In particular, we have shown that several aspects of the agent and of the RP are sensitive to features of the environment the agent has to adapt to.

We ran both simulations and real world experiments. Simulations have proved very useful to test general design criteria, and our methodology resulted robust enough to be portable from simulated to real worlds without major problems. The results obtained so far allow us to make a few claims:

- Animat-like interactions in simple environments can be practically developed through supervised reinforcement learning. Fairly complex interaction can be developed even with simple, reactive agents. In particular, behavior patterns that appear to follow a sequential plan can be realized by a reactive agent when there is enough information in the environment to determine the right sequencing of actions (see the Chase/Feed/Escape behavior in Section 6). However, the addition of nonreactive elements, like a memory of past perceptions, can improve the level of adaptation to the dynamics of the environment.
- The genetic algorithm can be exploited to enforce adaptation of a physical robot to its environment. In a modular architecture, both basic skills and coordination can be learned.
- To develop a situated agent, both explicit design and machine learning have an important role. In our approach, the main design choices involve: (i) the sensors, actuators and controller's architecture of the agent; (ii) the artificial objects in the environment; (iii) the sensors and the

logic of the trainer; (iv) the overall shaping policy. Learning is in charge of developing the functions implemented by the various modules of the agent's controller.

- In shaping the agent's behavior, the trainer can assume a reasonably high-level position, abstracting from the details of the agent's anatomy and concentrating on agent-environment interactions.
- A careful design of the agent's architecture can speed up learning. The designer should understand, at least at a coarse level, the dynamics of the interaction between the agent and the environment and the relationships among different basic behaviors. We demonstrated through an experiment the disastrous effects of a bad design (see Section 6, Figures 20 and 21).

At present, we feel that we have not wholly exploited the power of ALECSYS. In particular:

- Our Animat-like tasks make only soft requirements to the sensorial capacities of the real robot. A major concern of our future research will be to give the sensory apparatus some learning skills. In this way we hope to have the possibility to work with a more rich environmental information (at present our sensors can do only little more than giving binary information about the presence or absence of simple objects).
- Complex interactions in a non-Markov situation (see Whitehead & Lin, this volume) will require a richer memory mechanism. We are currently trying to exploit hierarchical architectures to obtain proper sequential interactions (i.e., sequential behavior patterns when the environment does not provide enough information for a correct sequencing of actions; see Colombetti & Dorigo, 1993).
- To develop more interesting interactions, we are currently moving to environments with richer dynamics. We are also considering the possibility of developing multi-agent, cooperative behaviors.
- Recent results in reinforcement learning and training (Clouse & Utgoff, 1992) suggest that the design of the reinforcement program, which currently requires a substantial designer's effort, could be replaced by direct interaction with a human trainer. In the future, this possibility will be compared with another interesting option, that is the description of the target behavior through some kind of high-level, symbolic language.

Our system has also a number of weak points; in particular, two of them must be highlighted:

(i) Our learning modules do not address the temporal credit apportionment problem: our RP only generates immediate reinforcements in response to the actions of the learning agent. We do not know yet whether our learning algorithm can manage tasks in which delayed reinforcement is a must. First results are contradictory (see Dorigo & Schnepf, 1993) and further research is needed. Clearly, this issue is fundamental for developing more complex dynamic behavior, beyond the present limits of S-R responses.

(ii) Quite a large amount of work is to be put in the architectural design. It is sometime sustained, for example by Koza & Rice (1992), that the effort put into architecture design plus the effort required to solve issues arising from the use of reinforcement learning, can be greater than the effort required to directly program the robot by hand. Nevertheless we believe that, at least until efficient ways to automatically generate good and working architectures are devised (and the approach taken by Koza & Rice (1992) seems to be promising), there is no way out: architecture has to be designed. It is often said that, and it is also our position, architecture is the result of a learning process on an evolutionary time scale, while behavior is the result of a learning process on a life time scale. Obviously, the results of the first learning process constrain the possible outcomes of the second one. We are mainly interested in life scale learning; but we also recognize the importance of putting not too much hardwired solutions in our systems. Comparing our work with that of Mahadevan & Connell, we somewhat relax their constraints on the architecture: although we decompose the overall task by design, we do not impose any structure on the coordination between learning modules. Coordination is learned, in the same way as basic behaviors are.

As a whole, we believe that our work shows the importance of learning to achieve a satisfactory level of adaptation between an artificial agent and its environment. Clearly, much further research is needed to understand whether our approach can scale up to a complexity comparable to the adaptive behavior of living organisms.

Acknowledgments

This research has been partially funded by a M.U.R.S.T. 60% grant to Marco Colombetti for the year 1992, by a grant from CNR - Progetto finalizzato sistemi informatici e calcolo parallelo - Sottoprogetto 2 - Tema: Processori dedicati, and from CNR - Progetto finalizzato robotica - Sottoobiettivo 2 - Tema: ALPI. We would like to thank Sridhar Mahadevan, Mukesh Patel, Hans-Michael Voigt and Robert Richards for helpful comments on a draft version of this paper. Graziano Ravizza designed AutonoMouse II. Franco Dorigo designed and built AutonoMouse IV. Franco Dorigo, Andrea Maesani, Stefano Michi, Roberto Pellagatti, Roberto Piroddi, and Rino Rusconi participated in implementing and debugging ALECSYS. They also ran many of the experiments presented in this paper. Emanuela Prato-Previde discussed with us several terminological and technical issues connected with experimental psychology.

References

Agre, P. E., & D. Chapman, 1987. Pengi: An implementation of a theory of activity. *Proceedings of the 6th National Conference on Artificial Intelligence, AAAI 87*, Morgan Kaufmann, Los Altos, CA, 268–272.

Agre, P. E., & I. Horswill, this volume. Cultural support for improvisation.

Arkin, R. C., 1990. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, **6**, 1-2, 105–122.

Barto, A. G., S. J. Bradtke & S. P. Singh, this volume. Learning to act using real-time dynamic programming.

Barto, A. G., R. S. Sutton & C. W. Anderson, 1983. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics,* **13**, 834-846.

Beer, R. D., this volume. A dynamical systems perspective on autonomous agents.

Beer, R. D., & J. C. Gallagher, 1992. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, **1**, 1, MIT Press, 92–122.

Booker, L., 1988. Classifier Systems that Learn Internal World Models. *Machine Learning*, **3**, 2-3, 161–192.

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

Brooks, R. A., 1990a. Elephants don't play chess. *Robotics and Autonomous Systems*, **6**, 1-2, 3–16.

Brooks, R. A., 1990b. The behavior language: User's guide. MIT A.I. Lab Memo 1227, April.

Brooks, R. A., 1991a. Intelligence without representation. *Artificial Intelligence*, **47**, 1-3, 139–159.

Brooks, R. A., 1991b. Artificial Life and Real Robots. *Proceedings of the First European Conference on Artificial Life (ECAL)*, MIT Press, 3–10.

Camilli, A., R. Di Meglio, F. Baiardi, M. Vanneschi, D. Montanari & R. Serra, 1990. Classifier systems parallelization on MIMD architectures. Technical Report 3-17, CNR, Italy.

Clouse, J.A., & P.E. Utgoff, 1992. A teaching method for reinforcement learning. *Proceedings of the Ninth Conference on Machine Learning,* Aberdeen, Scotland, 92–101.

Colombetti, M. & M. Dorigo, 1992. Learning to Control an Autonomous Robot by Distributed Genetic Algorithms. *Proceedings of From Animals To Animats, Second International Conference on Simulation of Adaptive Behavior (SAB92)*, Honolulu.

Colombetti, M. & M. Dorigo, 1993. Learning to plan. *Forthcoming*.

Compiani, M., D. Montanari, R. Serra & G. Valastro, 1989. Classifier systems and neural networks. In E. R. Caianiello (ed.), *Parallel architectures and neural networks*, World Scientific.

Dorigo, M., 1993. Genetic and non-genetic operators in ALECSYS. To appear in *Evolutionary Computation Journal*.

Dorigo, M., 1992a. Using transputers to increase speed and flexibility of genetics-based machine learning systems. *Microprocessing and Microprogramming Journal*, **34**, 147–152.

Dorigo, M., 1992b. ALECSYS and the AutonoMouse: Learning to control a real robot by distributed classifier systems. Technical Report 92–011, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy.

Dorigo, M., 1992c. *Optimization, learning and natural algorithms*, Ph. D. Dissertation, Politecnico di Milano, Milan, Italy.

Dorigo, M., & U. Schnepf, 1993. Genetics-based machine learning and behavior-based robotics: A New Synthesis. *IEEE Transactions on Systems, Man, and Cybernetics*, **23**, 1.

Dorigo, M., & E. Sirtori, 1991. ALECSYS: A parallel laboratory for learning Classifier systems, *Proceedings of Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Diego, CA, 296–302.

Grefenstette, J.J., C.L. Ramsey & A.C. Schultz, 1990. Learning sequential decision rules using simulation models and competition. *Machine Learning*, **5**, 4, 355–381.

Hammond, K. J., T. M. Converse & J. W. Grass, this volume. The stabilization of environments.

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

Holland, J. H., & J. S. Reitman, 1978. Cognitive systems based on adaptive algorithms. In D.A. Waterman & F. Hayes-Roth (Eds.), *Pattern-directed inference systems*. Academic Press, New York.

Horswill, I., this volume. Analysis of adaptation and environment.

Kaelbling, L. P., 1987. An architecture for intelligent reactive systems. In M. P. Georgeff & A. L. Lansky, eds., *Reasoning about Actions and Plans*, Morgan Kaufmann, Los Altos, CA, 395–410.

Kaelbling, L. P., 1990. Learning in embedded systems. Ph. D. dissertation, Stanford University, Stanford, California.

Kaelbling, L. P., 1991. An adaptable mobile robot. *Proceedings of the First European Conference on Artificial Life (ECAL)*, MIT Press, 41–47.

Kaelbling, L. P., & S. J. Rosenschein, 1991. Action and planning in embedded agents. *Robotics and Autonomous Systems*, **6**, 1-2, 35–48.

Koza, J. R., 1992. *Genetic Programming: On programming computers by means of natural selection and genetics*. MIT Press.

Koza, J. R., & J. P. Rice, 1992. Automatic Programming of Robots using Genetic Programming. *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI 92*, Morgan Kaufmann, San-Jose, CA.

Lin, L-J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, **8**, 3-4, 293–322.

Maes, P., & R.A. Brooks, 1990. Learning to coordinate behaviors. *Proceedings of the Eigth National Conference on Artificial Intelligence, AAAI-90,* 796–802.

Mahadevan, S., 1992. Enhancing transfer in reinforcement learning by building stochastic models of robots actions. *Proceedings of the 9th Conference on Machine Learning,* Aberdeen, Scotland, 290–299.

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

Millan, J. del R., this volume. Reinforcement learning of goal-directed obstacle-avoiding reaction strategies in an autonomous mobile robot.

Millan, J. del R., & C. Torras, 1992. A reinforcement connectionist approach to robot path finding in non maze-like environments. *Machine Learning*, **8**, 3-4, 363–395.

Piroddi, R., & R. Rusconi, 1992. A parallel classifier system to solve learning problems (in Italian). Master Thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy.

Robertson, G. G., & R. L. Riolo, 1988. A tale of two classifier systems. *Machine Learning*, **3**, 2-3, 139–160.

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.

Shepanski, J. F., & S. A. Macy, 1987. Manual training techniques of autonomous systems based on artificial neural networks. *Proceedings of the IEEE First Annual International Conference on Neural Networks*, San Diego, California, Vol.IV, 697–704.

Singh, S. P., 1992. Transfer of Learning by Composing Solutions of Elemental Sequential Tasks. *Machine Learning*, **8**, 3-4, 323–339.

Skinner, B. F., 1938. *The behavior of organisms: An experimental analysis*, D. Appleton Century, New York.

Sutton, R. S., 1984. Temporal credit assignment in reinforcement learning. Ph.D. dissertation, Dept. of computer and information science, University of Massachusetts, Amherst, MA.

Sutton, R. S., 1988. Learning to predict by the methods of temporal differences. *Machine Learning*, **3**, 1, 9–44.

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., 1991a. A complexity analysis of cooperative mechanisms in reinforcement learning. *Proceedings of the 9th National Conference on Artificial Intelligence*, 607–613.

Whitehead, S. D., 1991b. A study of cooperative mechanisms for faster reinforcement learning. Technical Report CS-365, University of Rochester, NY.

Whitehead, S. D., & D. H. Ballard, 1991. Learning to perceive and act by trial and error. *Machine Learning*, **7**, 1, 45–83.

Whitehead, S. D., & L. J. Lin, this volume. Reinforcement learning in non-Markov environments.

Williams, R. J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, **8**, 3-4, 229–256.

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