Behavior of an Adaptive Self-organizing Autonomous Agent Working with Cues and Competing Concepts

Csaba Szepesvári*
János Bolyai Institute of Mathematics

András Lőrincz†
Hungarian Academy of Sciences

A brain model–based alternative to reinforcement learning is presented that integrates artificial neural networks and knowledge-based systems into one unit or agent for goal-oriented problem solving. The agent may possess inherited and learned artificial neural networks and knowledge-based subsystems. The agent has and develops ANN cues to the environment for dimensionality reduction (data compression) to ease the problem of combinatorial explosion. Here, a dynamical concept model is put forward that builds cue models of the phenomena in the world, designs dynamical action sets (concepts), and makes them compete in a spreading-activation neural stage to reach decision. The agent works under closed-loop control. Here we examine a simple robotlike object in a two-dimensional conditionally probabilistic space.

Key Words: adaptivity; artificial neural networks; knowledge-based system; self-organization; activation spreading; autonomous system

Introduction

The history of developments in knowledge-based systems (KBSs) is lengthy (see, for example, Bundy, 1990), as is the history of research on artificial neural network (ANN) models, the latter dating back at least to the original work of Hebb (1949) (see, for example, Hertz, Krogh & Palmer, 1991). Ever since their universal approximator nature was proved (Hornik, Stinchcombe & White, 1989), there has been renewed interest in ANN systems. Just a few years ago, KBS and ANN were conceived of as different. Recent articles, however, report on ANN models that solve KBS problems (Peng and Reggia, 1989; Thagard, 1989). It is, in fact, difficult to make a clear distinction between KBS and ANN models. One might try to define KBSs as systems having a collection of if-then rules. However, the simplest neural system is a receptor neuron connected to a motor neuron and that similarly works as

* János Bolyai Institute of Mathematics, Attila József University of Szeged, Szeged, Hungary H-6720; szepes@obelix.iki.kfki.hu
† Department of Photophysics, Institute of Isotopes of the Hungarian Academy of Sciences, P.O. Box 77, Budapest, Hungary H-1525; lorincz@obelix.iki.kfki.hu

an if-then rule: If the stimulus is appropriate, then the receptor neuron activates the motor neuron, and the reaction to the stimulus is created.

One might try, then, to consider every ANN as a KBS. From this point of view, one might consider the perceptron as an if-then rule with a decision surface: If the perceptron's n dimensional input is above a decision hypersurface, then the output is high; otherwise, the output is low. Although one may approach a decision surface with KBS methods, the continuity of the problem does not fit efficient KBS procedures.

The problem can be approached from another direction: KBS are designed to and can handle the sequential rules of a simplified world. Even then, sequentiality has its own traps. With large problems, one meets combinatorial explosion in computation time or in memory requirements. Certain ANN systems, on the other hand, can avoid combinatorial explosion (Judd, 1990), but they are less efficient in handling sequentiality.

Our goal is the design of adaptive control agents. To this end, let us consider the game of chess as an example. There are 32 figures in chess, with approximately 100 possible moves in one iteration. Compare this with the image of chess on the retina, which has more than 1 million pixels (rods and cones), as well as color and intensity variations, and therefore a much larger degree of freedom. If we take the numerical example, considering six steps ahead, the number of variations is 100^6, and this can be compared to the degree of freedom of the recognition problem, which is on the order of $10^{1,000,000}$ assuming only a single color and a ten-grade gray scale. The processing time in chess, however, is not limited by the recognition of the chess table but by the recognition of the situation, the possible outcomes—in other words, the sequentiality. Sequentiality seems to have a different working mechanism in the brain, and we shall design it with the help of KBS methods.

When formulating a control problem, one often assumes that the input space on which the controller works is an explicit or accurate model of the system. Another approach is to assume that the model is accurate in a probabilistic manner: If S denotes the set of states of the model and A the set of actions, then for every s, t pair where s and t are states from S, there is a number $0 < p(s, u, t) < 1$, which is the probability of reaching t taking action u in state s. This control problem is the so-called Markovian decision problem. Most of the existing approaches, such as reinforcement learning (RL) (Widrow, Gupta & Maitra, 1973), Q-learning (Watkins, 1989), LRTA* (Learning Real-time A*) (Korf, 1990), and RDP (Real-time Dynamic Programming) (Barto, Bradtke & Singh, 1991) use this formulation. The algorithms, however, require a memory of the size of approximately at least $O(n)$, where n is the size of the state space, and that is hard to fulfill. There are suggestions in the literature to overcome this problem, such as the G-learning algorithm of Chapman

and Kaelbling (1991) that recursively splits the input space based on statistical measures of differences in reinforcement received, or the algorithms of Anderson (1987) and Lin (1990), who have successfully combined TD (temporal difference) (Sutton, 1988) methods with connectionist back-propagation. For further information, refer to Varela and Bourgine (1992).

Our starting point, or recipe, in the design of KBS and ANN integration is to use self-organizing ANNs as a means of compressing the tremendous information from the external world to a set of high or low bits. It is with this reduced set, this internal representation, that KBS could try to work by searching for rules, sets of rules, and sequences of rules.

The working mechanism we present here is an alternative to reinforcement learning. Our main concern about RL and versions of RL has two components: First, in a real world of competitive and collaborative autonomous agents, we can define conditional probabilities in general, but not probabilities, and second, we cannot give (or it is extremely difficult to give) the measure of goals or the cost of actions. There is, however, a natural building principle of autonomous agents that we intend to use, and this principle relies on the existence of dangers in the real world: Problems should be solved quickly to increase the spare time available for exploring the world. The more spare time the agent has to explore the external world, the less chance it has of falling into unexpected, cumulated, and thus unsolvable traps, and the more chance it has of survival. The attempts to solve problems simultaneously and in a time-saving fashion determines the problem (or goal) priority through the explored structure of events and problem hierarchy in a natural fashion. Goal priority will always be a function of the explored world and of the actual and detected state of the agent. Nevertheless, we do not try to establish goal priority, but we do try to create a framework where it is self-defined in a dynamical fashion (that is, the agent collects and connects experienced situations). In a given decision, the agent tries to build action sets (a set of concepts) based on these experienced situations, with the help of neural methods. The method used here is activation spreading (Collins and Loftus, 1975; Huberman and Hogg, 1987). In parallel, there is competition between concepts (Csányi, 1982), and the winning concept is selected (Grossberg, 1968) at time-out, leading to an action. The decision then influences the world and the events of the world, after which these changes are detected with the self-organizing ANNs, and they form new experiences.

The competition of concepts that is realized with the help of ANN methods through activation spreading and winning selection is closely related to Thagard's ECHO system (Thagard, 1989) and to Maes's behavioral models (Maes, 1992). Thagard considered competing consistent subsets of arguments in decision making. Maes made a decision between preprogrammed behaviors with an ANN network and has

shown that the resulting behavior selection is data-driven (opportunistic), goal-driven (the more a behavior contributes to goals, the higher its activation level), sensitive to goal conflicts, and has a certain inertia or bias (toward previous sequences). These properties may also be expected from our model.

2 Self-organizing Cues to the Environment: Dimensionality Reduction

If one says that a complex environmental stimulus has triggered a response, it means that (1) there is a complex environment that can and did stimulate the system, (2) there was a simple or complex action of the system, and (3) the action was triggered by the stimulus. The action may then be called *response*. The word triggered means that the response was launched, and it may be considered that the stimulus set to high a switch inside the system, allowing launching: That is, the environment steadily stimulates the system through its inputs, and the inputs are forwarded to switches that can launch responses if the stimulus matches the input system of the switch (Csányi, 1982). In Csányi's terminology, the switch is known as a *cue*. The output of the switch is an *internal representation* of reduced dimensionality of the external world that is set high if the key matches the stimulus or low if it does not. The output of the switch is the simplest representation of the world. If a switch is set high, then we say that a feature is present. If this feature triggers an immediate response, then we call it a *reflex*, and it could be an inherited property of the system. If it does not require an immediate response, then it may be used for further processing for later responses. This key concept of the external world—if the stimulus fits, then a switch is set high (Cloak, 1975)—is the place where a KBS can operate because the dimension is already reduced and there is time for processing.

The first building block of the artificial agent is, then, an input system that provides outputs of reduced dimensionality. The dimension-reducing system depends on the task one tries to solve. Next, let us look at a few examples.

If there are stable features in the external world then, for example, a categorizing system may serve well. The categories could be inherited or learned. In special cases, they could be both. If inherited and nonchangeable, then adaptivity is severely decreased. For category learning, only self-organizing ANNs are suitable because there is no supervisor who could pair inputs to outputs. Two self-organizing networks that warrant special mention are the two-layer ART (Adaptive Resonance Theory) network of Carpenter and Grossberg (1987), which sets up categories with the help of behavioral success, and the Hebb and anti-Hebb (HAH) network of Földiák (1991), which has a constant drive for distinguishing inputs and forms memory-saving sparse representation from them.

The example we shall be treating is defined on a two-dimensional grid. For this purpose, a self-organizing network that could build spatial filters is needed. One solution for creating such a network is given by Szepesvári, Balázs, and Lőrincz (in press). The object of the external world excites the receptors of a high-resolution "eye," and the neurons develop spatially localized filters. The system needs a categorizing system, as discussed earlier; the HAH network of Földiák (1991) is suitable. It may be interesting to note that the spatial filters also could be developed by using HAH principles (Fomin & Lőrincz, 1993). In fact, the HAH formulation is quite general: Most neural network paradigms can be given as an HAH net or an HAH net architecture (Fomin & Lőrincz, 1993).

The next task is to develop a system that can deduce rules, build rule structures and sequences of rules, handle contradictions (as contradictions may arise during learning or when the world changes), and work on the outputs of the self-organizing neural networks, these being the internal representation of the external world. What, then, is a rule?

Building Concepts of the KBS

Now we will describe the self-developing and self-building KBS. From the point of view of the KBS, the artificial agent is made up of four parts:

1. The dimension-reducing self-organizing interface system; the input interface of the KBS,
2. The KBS itself
3. A decision stage that determines which of the possible actions will be taken
4. The output interface of the decision stage, which activates the operators of the system.

Because items 2 and 3 are closely interlaced, we shall describe them together.

From the system's point of view, the world comprises two parts: the internal world, (i.e., the system itself) and the rest of the world—the environment. From the operational point of view, we need three categories for the internal representation of the world:

1. Internal representations of objects, whether internal or external. The input interface of the KBS provides this information in the form of bits. Bits then represent objects or features of objects. We shall call the set of bits of the internal representation at a given instant an *I state*.
2. Operators of the system. We assume that the system can influence the environment, can move, may eat, grasp, and so on. This assumption means that the system has operators associated with the interaction; thus,

it may have an operator for moving forward or for lifting its foot or opening its mouth or just for contracting a muscle. Every operator is associated with a state: If the state is set high, then the operator is activated, but if it is set low, the operator is inactive. We shall call the set of operators at a given instant an *O state*. The operators defined here are sometimes known in the literature as *actions*.

3. Goals of the system. These are special states that can be either high or low. They initiate search procedures in the *I* and *O* states to find routes to fulfill them. We might say that the system is in a neutral state or that it is hungry or that it is feeling pain, for instance.

The task of the KBS is to build a model of the events in the world, including the interactions of the system with the world. Such a model is constructed under the following principles: The *I* states are the ones on which the KBS should work. The task is then to experience *I-O-I* time sequences. These *I-O-I* symbols will be called *triplets*; they have an initial state, an operator, and a final state. Triplets that were encountered contain all the information the system can collect, and the task is to process this information to fulfill the goals. The place where systems might differ is their strategy for subtracting rules to simplify or speed up search procedures, to generalize from examples, and so forth.

It is worth mentioning that if one is collecting the *I-O-I* triplets for a given operator *O*, one cannot expect the influence of *O* to be predicted with certainty, and that holds for deterministic worlds as well. The reason for this is that the internal representation has restricted information about the whole world, because the input system may detect only a small part of the world and the reduced dimensionality (compressed) output of the input interface of the KBS filters out parts or important details. In mathematical form, operators are not mappings on the *I*-state space but relations.

The KBS builds up a directed graph of *I-O-I* triplets and associates goals with those triplets that have led to the fulfillment of goals. In a given situation, the goal or goals may be active; if so, spreading activation (Collins & Loftus, 1975; Huberman & Hogg, 1987) is initiated on the directed graph. Decision is made by choosing the operator of the largest activity triplet that belongs to the *kernel*, the kernel being the set of triplets that has an initial state identical or close to the actual state of the system.

These concepts are elaborated in the following sections.

4 List of Notions

The example we present here corresponds to a simplified robotlike object. For expedience, we shall call it *Robject*.

Goals are inherited bits of Robject that may be either high or low. If a goal becomes high, then a search procedure is initiated. The purpose of the search is to find an operator or a set of operators of Robject that could set the goal low (if Robject could fulfill the goal) according to its experiences. The task of this article is to describe the accumulation of experiences and the search procedure.

We shall use the expression *cognitive system* as an equivalent of the KBS from here on.

Internal representation or *internal state* (I) is a set of bits provided by the self-organizing dimension-reducing interface of the cognitive system about the external world and the internal state of Robject.

Internal bits of the internal representation (I^{int}) are the set of bits that contain information about Robject and parts of Robject. The possibility to distinguish between internal bits and the rest of I is considered an inherited property. This distinction is not necessary and was made to speed up search procedures in the model. If the distinction were undesirable, then one can think of ways to learn this set through pain, because pain helps us to learn about ourselves and, if so, then internal bits could be learned through experience. Consider a human defect, the inability of infants to feel pain. In ordinary cases, if a move results in pain, then the move is stopped. However, if there is no pain, then the move is continued. The lack of pain can lead to self-destruction: Such is the case in self-cannibalism in some infants. The self-cannibalism must be stopped in a forceful manner until one can find other ways of influencing the infant. Now, assume that pain is present and an internal visual representation has been formed previously. Then the pain and the visual subset of the internal representation that contains the move and the objects that take part in the move may be associated. If the algorithm can generalize, then experiences can lead to the substance or place of pain in the visual part of the internal representation, which is equivalent to learning ourselves in the visual representation. These associations between subsets of the internal representation could reduce the inherited set of internal bits.

Bits of the external world in the internal representation (I^{ext}) are the rest of the bits. *External world similarity in I* (S^{ext}) is the normalized inner product of the I^{ext} bits of two internal states.

The *input interface* is between the external world and the cognitive system of Robject. It is made of detectors and inherited or self-organizing systems (here, ANNs) that produce high or low bits as outputs and then input these to the cognitive system.

Operators of Robject $\{O_i\ i = 1, 2, \ldots, n\}$ are the set of bits to which Robject has access—that is, Robject is capable of setting these high or low and can produce some changes in the world.

A *bare triplet* consists of an initial internal representation of the external world (i.e., the initial internal state [*IIS* or I_i]), an operator O, and a final internal representation of the external world (i.e., the final internal state [*FIS* or I_f]). The triplet is ordered as *IIS*, O, and *FIS*. Bare triplets are denoted by small letters.

In *related triplets*, a given *IIS*, I_i, and an operator O may result in different *FIS*s: $I_i^{(n)}$ with a varying, usually growing number of n as experiences are accumulated. The fact that n may be larger than 1 is a consequence of changes of the external world as well as the fact that the internal representation realizes a reduction of the external world, and thus operators are not mappings but relations on internal states. Triplets and the set of experienced triplets that have identical *IIS* and O will be called *related triplets* and the *set of related triplets*, respectively.

The significance of a given triplet t in a decision-making procedure is small if the final state of the system is uncertain (i.e., if the set of related triplets is large). We shall associate a value to each triplet, the so-called *significance value* (SV), that is influenced by experiencing related triplets.

A *dressed triplet* consists of a bare triplet, t, and all the important information on bare triplet t, such as the SV of t and the goal or goals that are associated with t. Dressed triplets will be denoted by capital letters.

Short-term memory (STM) is a finite length (m), first-in, first-out (FIFO) buffer that contains a list of the last m dressed triplets. *Long-term memory* (LTM) is a weighted, directed graph with dressed triplets as nodes and directed connections provided by experience and read from the temporary buffer, STM.

Directed connections of LTM connect two triplets, T_1 and T_2, according to experienced time sequences, the first triplet's *FIS* being identical to the second triplet's *IIS*. The connection is directed from T_1 to T_2. Connections have weights of between 0 and 1. The stronger the weight, the more often T_2 is chosen after T_1. Weights are updated when triplets are memorized, which occurs when the triplet becomes part of the LTM.

Successor triplets of a triplet in LTM are the triplets that are in the LTM and follow the given triplet in time on the basis of experience.

Pain is an inherited experience predetermined in the simulation. Situations, operators, or operators applied in situations may lead to pain (i.e., pain may be associated with triplets). In other words, a triplet may result in pain. Triplets that result in pain are collected and memorized in the pain buffer. Conditions of pain are determined by the simulation. Triplets that are experienced as leading to pain may not be chosen at later stages in the present procedure (as will be discussed later). The model we are treating allows, however, for a more general treatment of pain because one of the goals is to avoid pain. Treating pains and goals on an equal footing would require treating pains as negative goals and would include those with a negative sign in our

activation-spreading model. At present, the system makes every effort to avoid pain. The preceding generalization, however, could consider goals with the (temporary) presence of pains.

A goal is associated with triplet t if (1) a goal is reached, (2) triplet t is in STM, and (3) triplet t is memorized.

Decision is an algorithm implemented by spreading activation, time-out, and winner-take-all mechanisms that results in choosing a triplet. The operator of the triplet is then applied: The bit that corresponds to the operator is set high.

Creating and Updating the KBS

There are different cases one should consider during the course of learning, such as (1) the very first experiencing step, (2) steps following decisions, and (3) steps between decisions. We start by assuming that a decision was just made and a new state was reached. The very first step of the learning procedure is simple in this respect: It starts with a random choice of an operator. In other respects, it is identical to the case when there is no acceptable solution (this is dealt with later).

First, we consider the adjustments after the decision, when the new situation was just detected, and it was just processed by the dimension-reducing interface and passed to the cognitive space as an *FIS*. The adjusting procedure to be described next will be called *maintenance*.

Let us denote the triplet that was just experienced as

$$t' = I_1 O I_2' \tag{1}$$

5.1 Maintenance of the STM

If operator O was chosen randomly, then triplet t' is communicated to STM. If operator O was not chosen randomly and the dressed triplet that was decided about was T and contained triplet

$$t = I_1 O I_2 \tag{2}$$

and $t = t'$, then T is communicated to STM. Otherwise t' is dressed and the dressed T' with information on pain and goals is communicated to STM.

5.2 Maintenance of the LTM

First, SV is updated. To do so, the set of dressed triplets is determined that have the same *IIS* and O as has T. Then the updating proceeds according to the *fading average method*:

$$SV(\text{new}) = \alpha SV(\text{old}) + (1 - \alpha) \tag{3}$$

if a bare triplet equals t' and

$$SV(\text{new}) = \alpha SV(\text{old}) \tag{4}$$

otherwise, with $0 < \alpha < 1$: That is, the SV is either increased toward 1 or decreased toward 0, depending on whether the experience is consistent with the triplet.

In the second step, parameter AGE is upgraded. The role of AGE is to distinguish between frequent and rare events and not to allow the fading of important but rare information. It is similar to postsynaptic learning. AGE is a common parameter of the set of triplets that share the same IIS. A triplet set that shares the same IIS will be called the AGE *group* of IIS. For every t that belongs to the AGE set of t',

$$AGE(\text{new}) = AGE(\text{old}) + 1 \tag{5}$$

If LTM is upgraded by adding a new triplet to it (this procedure will be described later), then every triplet of the AGE set that belongs to the new triplet will have zero AGE.

5.3 Maintenance of Connection Strengths

The first step is to remember whether a triplet resulted in pain; moreover, one must remember the triplet and its route if a goal was reached. If triplet t' resulted in pain, then t' becomes a member of the pain group. If a triplet fulfilled a goal, then STM is transcribed to LTM by extending the LTM chain: One single triplet is connected to the appropriate chain if not all the triplets of STM are in that chain yet. The connections between the elements of the LTM chain that corresponds to the STM are increased by unit strengths. In the simulation, one unit is 0.1. Care is taken that connection strengths should range between 0 and 1. Elements of the chain are associated with the goal.

From time to time, the memory itself is updated. This procedure is called *memory fading*, and is considered later in this article. Every memory updating begins with the investigation of AGE. If AGE is larger than a threshold (old), then all the connections that start from the same AGE group are decreased by a unit. The fall of connection strengths may lead to triplets with no connections. Such triplets are handled in the memory-fading procedure.

6 "Thinking" of Further Actions

In the thinking procedure, concepts toward future actions are designed. The FIS of t' becomes the actual internal state (AIS or I_{act}). First, the dangerous triplets of the AGE group of the AIS are collected. Then the set of goals is determined. More than one goal may be present. The goal set is inherited. The conditions when a goal is activated or deactivated are preprogrammed.

The next step is the initialization. We start with initialization of the kernel, which is that collection of triplets that can be decided about as the present state that may apply, the operator to apply, and the final state to be reached. The kernel shall include those triplets that are similar to I_{act} in their external bits—that is, triplet $t = I_1 O I_2$ shall be included in the kernel if

$$S^{ext}(I_1, I_{act}) > \theta \tag{6}$$

where θ is the kernel threshold. We have tried small θ values to include triplets that could be different from I_{act} in one or more bits. However, in the examples we considered, there was no improvement in generalization and the speed of computation was decreased. In some cases, it resulted in confusing, very different situations. This failure is considered to be the result of a low number of features in our simulations. It seems important to build feature detectors that could generalize situations and are not misled by unimportant details. We believe, however, that it is not the internal representation where this approach is to be applied because:

1. If it could lead to significant improvement, then it is rather the interface that should be modified to produce better features of the external world.
2. Important features are specified not by the internal representation but by the goals: It seems attractive to build self-organizing networks that collect features of the initial states of the triplets which lead to the fulfillment of goal(s). Correlations between features of *IIS*s of those triplets that reached a goal and the goal itself can be used to define subgoals in a self-organizing fashion. This point is under investigation currently.

Other conditions for choosing a triplet for the kernel are that the actual goal(s) should be associated with it and the operator of T should be applicable in the given *IIS*. Under the conditions we are using, those cases when O of T is not applicable are simple; an example might be if pain is overriding the decision. More complicated situations can arise if the triplets of the kernel are chosen with larger freedom.

We continue in different ways if the present situation is the result of a true decision and not just the result of a random choice between operators. If the previous step was just an operator applied randomly, then the initialization of the kernel is finished at this point. If true decision making was the previous step, then the kernel will contain other triplets as well: The decision was made in favor of an operator of the winning triplet and that triplet has directed connections toward the goal(s). Successor triplets of the winning triplet are included in the kernel if their operators are applicable.

For quick orientation, it should be noted that in the numerical procedure (1) excitation shall be given to the actual goals, (2) stimuli will be given to some triplets that have the present *IIS* and fulfill other conditions, (3) the excitation will be allowed

to spread between triplets on a restricted set of the whole directed graphs—on the extended kernel—starting from the goals and going in the opposite direction, (4) excitation can reach the kernel, and (5) members of the kernel accumulate excitations. The sum of the cumulated excitation and the stimuli are the subjects of the winner-take-all competition.

The next step is to compute the stimuli to the members of the kernel. The stimulus, S_i of dressed triplet T is determined by multiplying T's relation to goal (RTG) and T's usefulness (UF). If T can fulfill the goal, then the value of RTG is 1; otherwise, it is 0.5. The UF value is a function of the difference of similarity (S) and danger (D):

$$UF = f(S - D) \tag{7}$$

where function f (1) is 0 if its argument is 0, (2) has a slope of 45 degrees for negative arguments, and (3) has a slope of SV for positive arguments. S is given as

$$S = \frac{S^{\text{ext}}(I_1, I_{\text{act}}) - \theta}{1 - \theta} \tag{8}$$

I_{act} is the actual IIS, and θ is the kernel threshold. In the deterministic world examples, the value of θ was 0.99.

Danger relates to the question of whether triplet T could lead to pain. This is the very point that should be changed if pain is to be treated as a negative goal: That procedure could start by giving negative excitation (i.e., inhibition) to those triplets that resulted in pain in previous experiences. The inhibition then spreads in the same way as it does for goals. The present procedure is, however, somewhat different: Pain does not enter as a negative goal, but triplets that lead to pain are disconnected from LTM and are communicated to the pain buffer. The way to estimate the danger value of a triplet is to investigate the triplet buffer. We make a list (L) of I-O-I triplets of the triplet buffer, where the IIS is closer than θ_D (equals 1 here) to AIS measured by the S^{ext} similarity. Then we give the danger value D of an operator O of triplet t as the quotient:

$$D(I_{\text{act}}, O) = \frac{1 - SV(t)}{|L|} \sum_{I_1 O_1 I_2 \in L, O_1 = O} d[S^{\text{ext}}(I_1, I_{\text{act}})] \tag{9}$$

where $SV(t)$ is the significance value of the triplet under consideration and d is a nonlinear function giving 1 if its argument is 1 and 0 for argument 0. It was implemented as a sharp sigmoid function with threshold value near to 1. In other words, the summation is allowed for initial states of approximately 1-bit distance from the I_{act}. Let us consider the effect of Equation 9 in the extremes: (1) If $SV(t)$ is small, then it measures the frequency of operator O in the pain buffer; and (2) if

$SV(t)$ is near 1, then the danger value approaches 0 independent of the frequency of operator O in the pain buffer. The second extreme is suitable, because $SV(t)$ tells us how deterministic is the application of the operator O in state I_{act}, and the *FIS* of a triplet in the LTM may not contain pain.

This ends the determination of the external input, or stimuli (S_i), to the members of the kernel. There is also another threshold value associated with the initialization of the activation-spreading procedure, and the activities of members of the kernel shall be compared with it, namely the kernel threshold value, θ_k.

The next step is the extension of the kernel to all m-step successors of triplets in the kernel. Number m gives the depth of thinking. If, as may happen, there is no directed path that would connect any of the triplets of the kernel to the actual goal(s), then the member of the kernel having the largest S_i value is chosen, that value is compared to θ_k, and the operator of that triplet is chosen if S_i exceeds θ_k. Otherwise, a random choice of operators is made from the allowed set of operators. There is no need to separate this case, however, as the spreading activation model provides the same result.

Up to this point, we have recalled possible routes to achieve the goal. The choice of actions could be fairly poor or rich depending on the problem, the learning stage of the system, and the parameters of the system, such as threshold values, length of the STM, and the like. In any case, we are in a position to make a decision that will be strengthened or weakened by external feedback.

Activation Spreading and Memory Fading

Activation spreading is made opposite to the directions of the connections, under the following conditions:

1. Actual goal(s) receive activities at and for the first moment of the spreading process.
2. Members of the kernel cannot pass activities but accumulate the spreading activities.
3. Spreading proceeds according to the following update rule:

$$a_i(\text{new}) = \max[0, \gamma a_i(\text{old})] + (1 - \gamma)(S_a(I_N - \theta_a) - \delta) \tag{10}$$

if the i^{th} node is not part of the kernel, and

$$a_i(\text{new}) = a_i(\text{old}) + I_N \tag{11}$$

if the i^{th} node belongs to the kernel. I_N may be given as

$$I_N = \sum_{j \in (\text{successors of } i)} S(c_{ij}) a_j(\text{old}) \tag{12}$$

where c_{ij} is the connection strength increased by the memorizing procedure from STM and decreased by *AGE*, S is a nonlinear function, taken as a sigmoid in the computations, and the successor relation is given by the directed graph itself. Sigmoid S is important for accumulating learning and forgetting. Max takes the largest element of its arguments, S_a is the spreading activation sigmoid function with threshold θ_a, and δ is a loss factor. At every pass of an activation value to another node, there is a loss in the sum of activities. The loss factor is included to make the search more efficient for shorter paths to the goal or, in other words, in favor of shorter arguments.

Activation is spread up to time-out. At time-out, activations in the kernel are compared and the largest activation node is chosen. If that activation does not exceed θ_k, then a random choice between allowed operators is made. If the activation exceeds θ_k, then we say that there is enough reason to choose this triplet. Nevertheless, we have included another threshold in the procedure that can continuously lead the model to a random choice machine. This threshold is called the *random reason threshold* and is denoted by θ_r. Before every decision, a random number is generated between 0 and 1. If the random number exceeds θ_r, then again a random choice between allowed operators is made. If the random number doesn't exceed θ_k, then the largest activity node of the kernel is chosen and the operator of that node is set to 1. $\theta_r = 0$ corresponds to the random choice machine. In a number of runs, we have experimented with θ_r values that were continuously decreased to 0 as experiences were collected. θ_k and θ_r shall be called "enough reason" threshold (ERT) and random reason threshold (RRT), respectively.

It is important to let insignificant information fade and be forgotten. This is the last step of the procedure. It is regularly done after every twentieth trial in the experiments conducted. Memory fading means leaving out triplets from the LTM. Triplets fall out of the LTM if the triplet significance value decreases below a threshold or if the triplet no longer has any connections.

8 Parallel Properties of the Model

This section is devoted to the run time estimation of the system. The model was designed so that it may be run parallel at every level. We start by assuming that neural hardware is available and the processing time of the input interface is negligibly small (Agranat and Yariv, 1987; Agranat, Neugebauer, and Yariv, 1990).

The first step of kernel initialization is to find those triplets that will build the kernel up. If it is a serial computer, then this step is proportional to n, where n is

the number of triplets. It is a one-step procedure for parallel architecture, as every triplet can make the decision whether or not it fits the conditions.

The second step of kernel initialization is the determination of the initial activity values of triplets in the kernel: The step length is limited by the search in the pain buffer. For the serial case, it is proportional to the size of the kernel and the size of the pain buffer. For the parallel case, this is a one-step procedure again, as triplets of the kernel may be compared to triplets of the pain buffer simultaneously.

The next steps are the extension of the kernel and the activation spreading. For serial architecture, both steps are $o(n \times m)$ long, where m is the possible length of kernel extension limited by the time-out. In case of parallel architecture, the computation time can proceed in depth step-by-step in a parallel fashion. That is, the computation time is proportional to m.

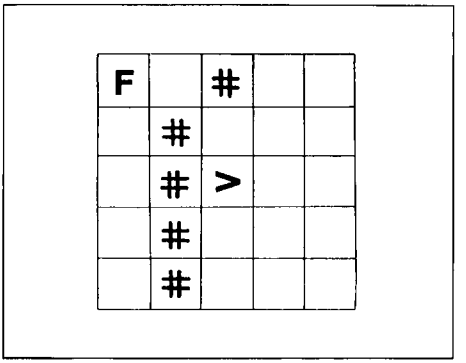
In conclusion, the run time is long on a serial machine but shortens considerably for parallel architectures. In fact, computation time for parallel architectures is of $o(m)$ duration, where m is the maximum length of learned chains of the directed graph (i.e., the depth of thinking).

Implementation of the Model

The example we present here corresponds to our simplified robotlike object. The adaptive part of the cognitive system and the decision system are independent of this special example. For Robject, we simulate the environment, consisting of objects causing pain and objects allowing energy refilling. It is easy to define *hunger* in this case: Robject is hungry if its energy is low, and that can initiate search procedures; Robject searches for ways to remove pain that can be formulated in the same way.

We are in the position to assume that Robject has well-developed self-organizing eyes (Szepesvári, Balázs, & Lőrincz, in press). The system is, however, more general; the network developed in the work mentioned or in the HAH formulation (Földiák, 1991; Fomin & Lőrincz, 1993) searches for independent high-order correlations in the input space and provides sparse code as outputs if those exist. Here, the search for correlations in the input space leads to the development of spatial filters of approximately the same size and shape. That is, we may assume, that some outputs of the dimension-reducing interface correspond to spatial regions. We shall assume that these correspond to a grid. We further assume that the input from every spatial region may be forwarded to a self-organizing categorizing system that can categorize the type of inputs in the regions. For self-organizing category learning, see, for example, the ART model of Carpenter and Grossberg (1987) or the sparse representation of Földiák (1991). Networks that are capable of accomplishing the categorization of

Figure 1
Robject's world. Epoch no. 1, trial no. 7. Eye: F-#—#—#—#—#—. Energy 0.90, mouth 0, pain 0. Operators: suck 0, forward 1, backward 0, turn left 0, turn right 0. Goals: stop hunger 0, remove pain 0. Robject is represented by the sign >, meaning that Robject is at grid point $x = 3$, $y = 3$ and is facing right. Robject's mouth is at the leading edge of the face (not shown). The suck operator should be high to acquire food. Another condition for eating is that the food should be in front of Robject. Food is denoted by *F*. If eating is successful, then energy is refilled to 1.00. In other instances, energy is constantly decreasing. Robject can move forward or backward on the grid and can turn left or right. Robject cannot enter a grid point if it is occupied by food or by a blockage (#). Any trial results in pain. Robject's eye is a "chameleon" eye; Robject can see its environment and is at the center of the environment.



poorly positioned objects have been presented by Olshausen, Anderson, and Van Essen (1993), Fukushima (1992), and Fomin and Lőrincz (1993).

The model we developed is somewhat more restricting in its assumptions, but it approximates the present possibilities of self-organizing neural networks if combined with inherited movements, such as eye movements and motions. We assume that Robject is living on a grid, and any object of the external world is placed on a certain grid point, as is Robject. The world is considered slow compared to Robject's motion; in other words (with one exception), the objects are static. At any grid point, only a single object or Robject may be present at any one time. Robject is watching the world through self-organized spatial filters. These spatial filters match the grid size of the external world, and Robject can see neighboring positions. To every position, there belongs one of three categories: the empty position, food at that position, or a certain blockage at that position. These categories are just bits without meaning for Robject when the simulation starts.

Robject's world is shown in Figure 1. Robject is represented by the sign >. It means that Robject is at grid point $x = 3$, $y = 3$, and is facing right. Robject's mouth is at the leading edge of the face (not shown). The "suck" operator should be high to acquire food. Another condition for eating is that the food should be in front of Robject. Food is denoted by *F*. If eating is successful, then energy is refilled to 1.00. In other instances, energy constantly is decreasing. Robject can move forward or backward on the grid and can turn left or right. Robject cannot enter a grid point if it is occupied by food or by a blockage (#). Any trial results in pain. Robject's eye

is a "chameleon" eye; it can see its 5×5 (or sometimes 3×3) grid environment. Robject is in the center of the environment, and the environment is represented by a vector of 24 components. Robject has two goals, stop hunger and remove pain.

The strongest assumptions were made about the operators (or actions) of Robject (i.e., the assumption that forward or backward motions match the grid size and left or right turns are always of 90 degrees). Operator fusion, which will be discussed later, could lead to a self-organized development of such operators. Work is in progress along these lines.

Experimental Results

In the numerical experiments that we ran the space was a 5×5 grid. The world was either just this 5×5 grid or could have periodic boundary conditions: Leaving the grid to the right, one enters from the left side, and similar conditions hold for left, up, and down. This case will be referred to as a *torus*. Different problems were treated: (1) problems in an empty space, where there was no blockage, (2) problems with blockages, and (3) a problem with moving (escaping), conditionally probabilistic food. For every experiment, we tried to determine the optimal solution and the results of a random machine. That was not always possible. Experiments were run in a way similar to the reinforcement example (Barto, Bradtke, & Singh, 1991): learning trials are organized into epochs, and one epoch has 20 trials. In every trial we used a time-out (this time-out is different from the time-out of activation spreading): If the experiment were not finished in the number of steps given by the time-out, then the experiment was stopped. Time-out is needed, because in many cases the problem may fall into traps and may not be capable of finishing the task (Barto, Bradtke, & Singh, 1991). After every epoch, 100 experiments were conducted randomly to test Robject's knowledge.

Consider first the experiments in empty space. The goal of these runs was to determine the effects and sensitivities of model parameters. If the food is placed at the center, then Robject has 24 possible positions and four possible directions from which to start. Taking into account the symmetries of the system, the actual number of different initial states is 32.

10.1 The *AGE* Parameter

Figures 2 and 3 present the result for different *AGE* parameters. Robject was started from different positions. The curves of Figure 2 show the average step number (action) Robject needed to eat the food. The optimal value is 2.4. Depending on the random initial positions, Robject had the chance to perform sometimes better than optimally. Random search would have the average of 47.8 steps.

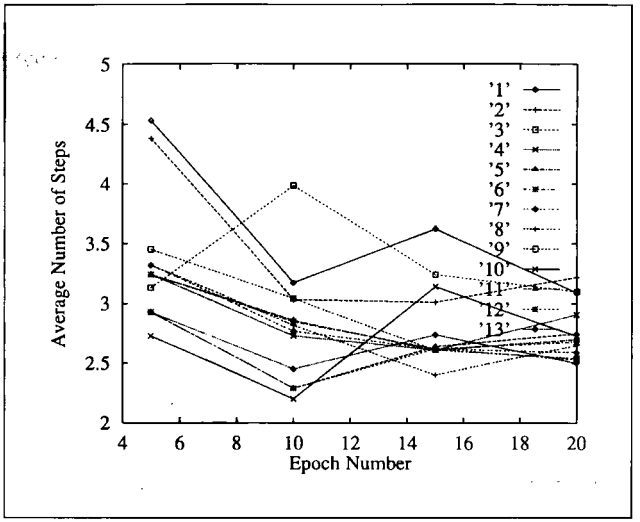


Figure 2
AGE curves. Different curves correspond to the course of learning for different *AGE* parameters. The average number of steps Robject needed in the test experiments to eat the food is given as a function of the epoch number. The *AGE* values of the curves range between 1 and 13 and are given at the right-hand side of the figure for every graph.

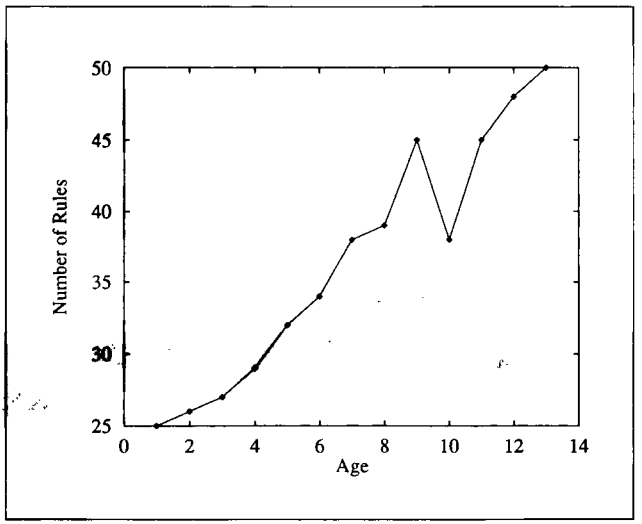


Figure 3
Number of rules as a function of the *AGE* parameter.

The trend of the graphs shows that the *AGE* parameter is important. Leaving the *AGE* out corresponds to 0 value (not shown). With increasing *AGE* up to 4, the performance, or the speed of learning, improves. Above that value there is no significant difference between the curves. On the other hand, the number of rules grows steadily with *AGE* according to Figure 3. Fifty rules for this problem is definitely too large (i.e., Robject retains more important and less efficient information). This,

however, may lead to traps (i.e., to infinite loops, when Robject moves in circles [as will be shown later]).

10.2 STM Length

The role of STM is to build the directed graph from the goals backward. In its present form, however, it limits the problems Robject can solve. The limitation comes from the fact that the longest directed path Robject can build is determined by the length of the STM, at least in simple problems, such as the empty space problem. The trivial solution would be to allow for a longer STM. That, however, is not desirable because, just like parameter *AGE*, a long STM can lead to the memorizing of complicated solutions and may eventually lead to a large number of traps. There are at least three ways to solve the problem:

1. Realistic problems are complex: The problem hierarchy itself can lengthen the directed graph. This is, however, an unintentional possibility that depends on the actual problem.
2. If the system itself could create subgoals and thus a goal hierarchy, then complex problems will be broken into simpler ones. That route seems feasible through feature extraction from triplets that satisfied the goals and thus the subgoals at later stages. This work is in progress.
3. Operator fusion is another possible route. If a given set of actions leads to the fulfillment of a goal, then one can try to include the action set itself into the set of operators. In this way, the same STM can correspond to much longer, more sophisticated actions.

Figure 4 shows the results for different STM lengths. If the STM is shorter than 5, then some of the problems become unsolvable for Robject. Results for those parameters are not shown in the figure. Beyond that thresholdlike effect, the performance is a smooth function of STM length within the limits of experimental error for this simple example.

The performance of the model is a smooth function of the rest of the parameters. We consider it an attractive property of the model, a sign of the approach's robustness.

10.3 Random reason threshold

RRT is the threshold value that determines whether the result of spreading activation at threshold is taken into account or a random operator is chosen. In these runs, the initial value of θ_r was different. In each case, θ_r was increased linearly to 1.0 for the end of the twentieth epoch. As may be seen in Figure 5, there is no strong trend in the results. The 0.1, 0.2, and 0.4 initial value curves learn less quickly, but the 0.3 initial value curve is fairly good at the beginning. Low RRT values learn less quickly

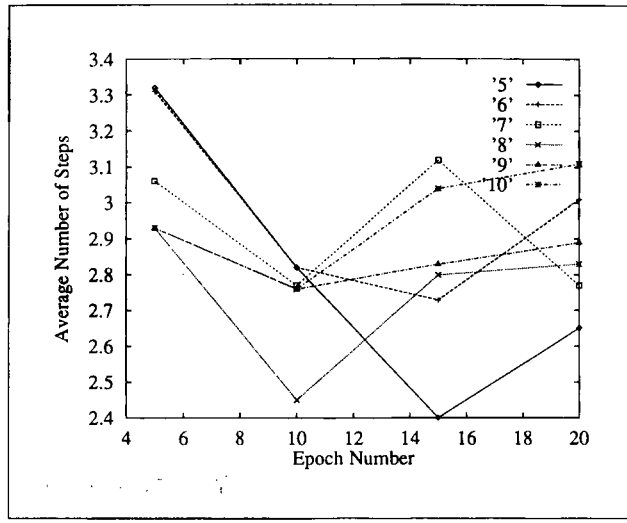


Figure 4
Dependence on the STM length. Numbers in the figure give the STM length for a given experiment. STMs shorter than five steps cannot solve the whole problem.

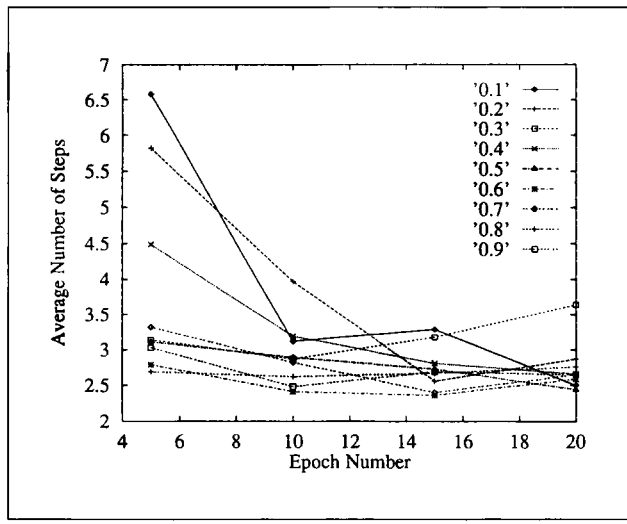


Figure 5
RRT curves. Different curves correspond to the course of learning for different RRT parameters. The average number of steps Robject needed in the test experiments to eat the food is given as a function of the epoch number. The RRT values range between 0.1 and 0.9 and are given at the right-hand side of the figure.

in general. The behavior of the 0.3 initial value curve is different from these: It starts with good results but deteriorates as time passes. There are many reasons why this might have happened, the most probable being that it accidentally found a relatively good solution early on but then had no chance to build up almost as good solutions. One good solution without a family of good solutions could be more fragile during the course of learning. The rest of the curves behave in a very similar fashion (i.e., the curves are equal within the experimental error for initial values larger than 0.4).

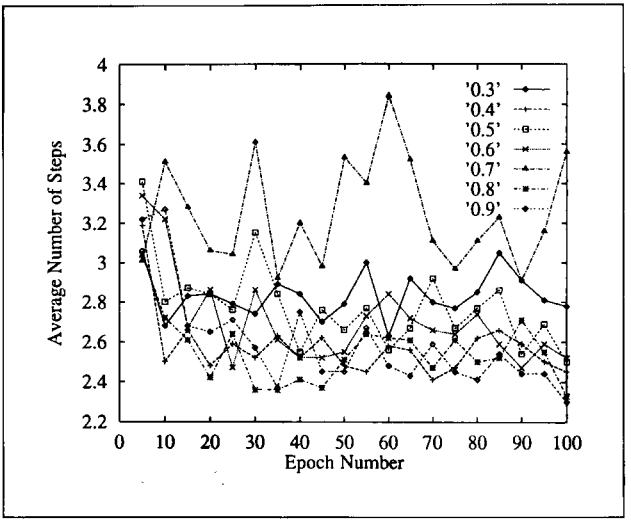


Figure 6
ERT curves. Different curves correspond to the course of learning for different ERT parameters. The average number of steps Robject needed in the test experiments to eat the food is given as a function of the epoch number. The ERT values range between 0.3 and 0.9 and are given at the right-hand side of the figure.

RRT was put into the model to provide a parameter that could play a role similar to the temperature of simulated annealing. This simple example, however, is not suitable for this test.

10.4 Enough reason threshold

The ERT value is the threshold value to which the results of activation spreading are compared. A high ERT value should lead to the cumulation of knowledge and to trial of a rule or rule set that was experienced many times. Low ERT values allow for the trial of less certain rules.

The results with different ERT values are given in Figure 6. The results are very similar to one another. The highest curve that belongs to the 0.7 ERT value shows that there is only one conclusion that can be drawn: The exploration parameter of these curves is not high enough to allow for escaping from a nonoptimal rule set. With the rules Robject has for the 0.7 ERT value, it can solve the problem, and that knowledge prohibits the finding of the optimal solution.

If the ERT value were 1.0, then we would have the random-choice machine, as the spreading activation cannot provide enough reason to make any choice. These curves, however, show that there is a broad region where ERT dependence is shallow.

10.5 Probabilistic Space

Figure 7 shows two possible positions of Robject in a 5×5 torus with four blockages. Robject can see only part of the world; in the simulations, Robject had a 3×3 eye. The shaded area corresponds to the region that Robject can see. The food was

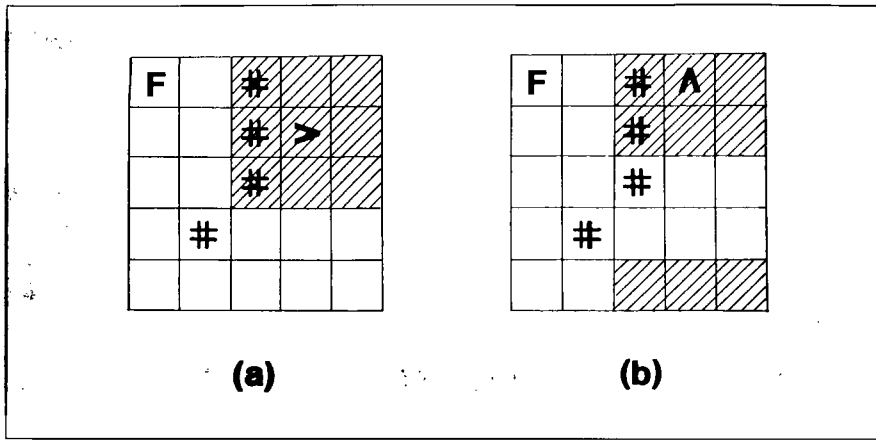


Figure 7
Probabilistic world. Two positions of Robject are shown for the 5 × 5 maze with a 3 × 3 eye. The shaded area corresponds to the area Robject can see. The # sign denotes blockages. The world is a torus: Leaving at one side, Robject enters again from the opposite side.

always placed to the upper right corner. The random-choice results averaged to 26 steps. This should be compared to the time-out value of 30. In other words, in a few cases Robject found the food just by random choice, but in most cases it could not. According to other runs (see, for instance, the next example), the average random-choice value if there is no time-out could be an order of magnitude higher.

This world could be considered a probabilistic world, as there are equivalent positions that Robject cannot distinguish. According to the experiments (Fig. 8) Robject could fall into traps in this example. Traps are problems for other methods too (see, for example, Barto, Bradtke & Singh, 1991). Robject could mix two solutions that corresponded to different positions and directions because certain states looked the same for Robject. This problem will be facilitated in more realistic examples, because fine details can distinguish the difference between similar cases. Working along this line, one might like to introduce a vigilance parameter, such as that of the ART model (Carpenter and Grossberg, 1987), that could raise or lower the degree of attention according to environmental feedback.

Another possibility to avoid traps that would fit Robject's architecture is to reduce the initial activation values of triplets in the kernel if the same triplets are in STM: The short-term memory offers that option and can help to avoid loops shorter than the length of STM.

There is a sharp peak in the curve: Here Robject built a trap and could not eat the food if it approached it from one direction. The RRT value, which was kept

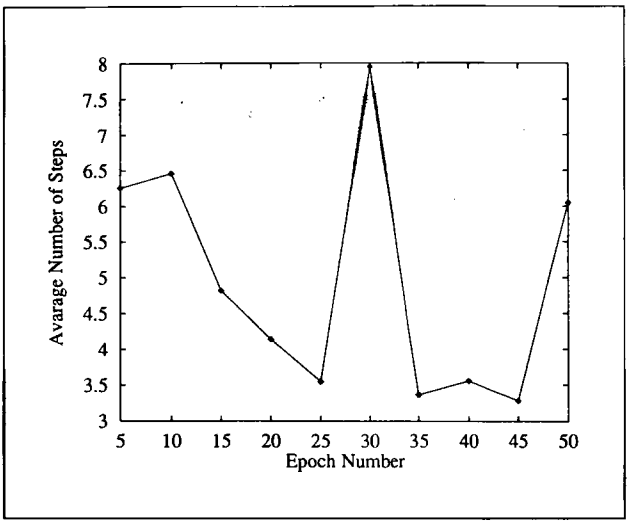


Figure 8
Average number of steps for the probabilistic world.

constant during these runs and allowed for random choice of operators in a few cases, allowed Robject to escape that trap.

10.6 Long-maze problem

The long-maze problem is shown in Figure 9. Two cases were tried—the full problem with and without pretraining. In this case, the problem is not on a torus but on a closed area. The full problem is 13 steps long. Robject is in the upper right part of the area and is facing right. The area is surrounded by walls, and there is a wall in the middle of the area as well.

Robject's development during the course of learning is shown in Figure 10. The time-out value was 75 for the experiments with the full problem. Curves belonging to this problem are denoted by small letters. These curves first slowly fall below the 75 value. The random-choice average with the same time-out was 74.976. After some trials—beyond more than 20 epochs—there is a sudden drop in the average step numbers in the tests: Robject cumulates knowledge that is suddenly enough to solve the problem. The sudden drop, however, is at very different positions; in the ten different runs, it ranged from 20 up to 45 epochs.

In another set of runs, we tried to start Robject from the lowest middle grid point first. The optimal solution to this problem takes six steps, and we allowed Robject ten epochs to learn it. Of ten experiments, six were successful and found the optimal six-step solution; in one experiment, it learned an eight-step variant of this pretraining example. These seven experiments were then tried in the full problem. Only three succeeded during the ten epochs we allowed for learning. These curves are denoted

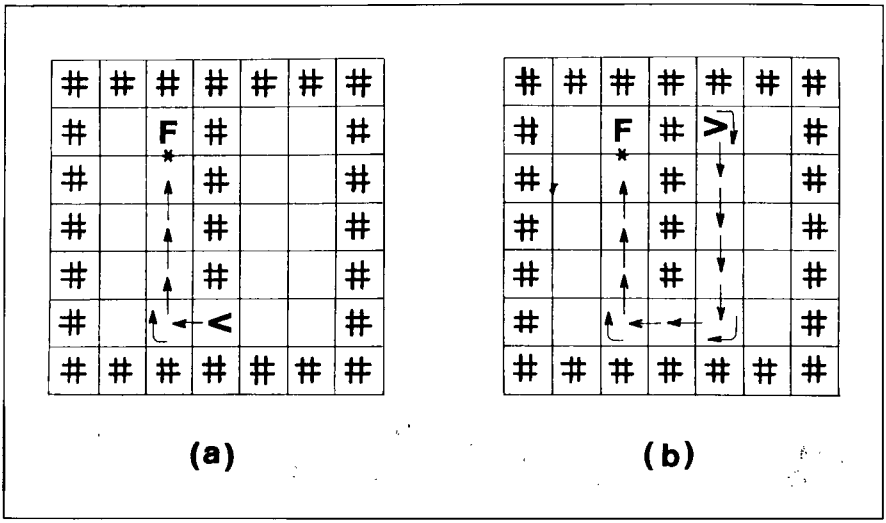
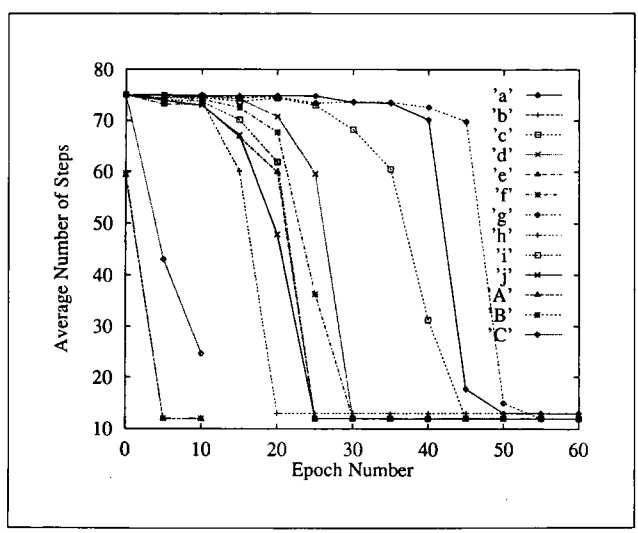


Figure 9
Long-maze problem. Parts (a) and (b) correspond to the pretraining and the full problems, respectively. In both cases, the initial states and the shortest solutions are shown. Curved arrows correspond to turning. The direction of the head of the curved arrows shows the direction of Robject after the turn. The world is a closed world, with blockages at the sides.

Figure 10
Average number of steps for the long-maze problem. Capital and small letters denote curves with and without pretraining, respectively. Time-out is 75. Random search result for the full problem is 74.976. Initial situations for the pretraining case and the full case are shown in curves (a) and (b), respectively. Pretraining happened with ten-epoch training. Seven of ten trials were successful. These seven cases were then tried for the full problem in a ten-epoch training and with the same time-out (75). Only three succeeded. These curves (two on top of each other) are at the left of the figure.



by capital letters and are the curves with the sharp drops at the beginning. Two of the curves are almost identical and are on top of each other. In the other cases, Robject fell into traps and could not find the solution within the ten-epoch training.

These examples indicate that a hierarchy of problems helps to solve more sophisticated ones. We estimate that the full problem from every position could be learned in approximately 30 steps with a well-designed pretraining set.

10.7 Conditionally probabilistic space

The problem of conditionally probabilistic space is a simple example of the type where Robject itself influences the position of other objects. In this special case, the food (prey) tries to escape from Robject. It does so in a way that discovers whether Robject makes a move and from then onward, it repeats every move of Robject with one step delayed, if that move is possible. The experiment is run on the torus. If the world is empty and if the original distance between Robject and the escaping prey is more than one block, then Robject has no chance to eat the prey. To avoid that case and make the problem solvable, one blockage was put into the torus and neither Robject nor the prey can make a move if that move is prohibited by the presence of the blockage: Standing at the left side of the blockage and facing right, the move-forward operation cannot be executed (and that trial will result in pain for Robject and sets the goal, remove pain, to high).

When the experiment was begun according to the lower left box of Figure 11, with Robject in the lowest left corner, the blockage in the top middle point, and the prey at position $x = 1$, $y = 1$, then Robject could not find the solution with a time-out value of 10. Results up to 50 epochs are shown in Figure 12. This experiment corresponds to the dotted curve with the xs and is denoted by S. Then we tried to pretrain Robject, starting from the $x = 0$, $y = 2$ position looking downward. Results show a slow decrease up to the fifteenth epoch, then Robject finds the solution and learns it quickly up to the optimal one. Results are shown by the S_1 solid curve of Figure 12. The best and the second-best solutions are shown in the top line of Figure 11: One is a four-step solution (plus the suck operation), the other a five-step solution, including the application of the suck operator. The random search result was 29.4 for a 30-step time-out (i.e., the random search average time for the unlimited time-out case is still much larger).

In another set of experiments, Robject was placed at the $x = 0$, $y = 3$ point and was looking upward. In these experiments, Robject had the knowledge from the previous experiments. The best (five steps) and the second-best (six steps) solutions are shown in the middle row of Figure 12. Robject was vacillating between these solutions. The results of these runs are shown by the S_2 curve (short dashed line with + signs) in Figure 11. Sometimes Robject's performance decreases but easily

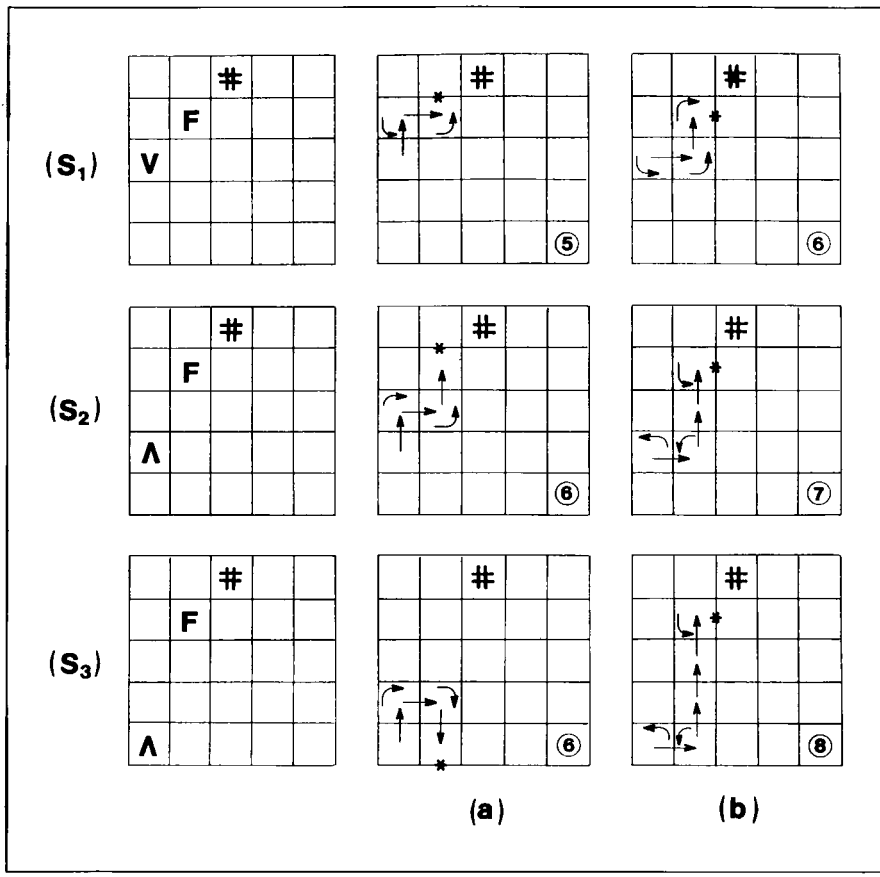


Figure 11
Nonprobabilistic world. Experiments with moving food (prey). The prey detects Robject's motions and repeats those when possible with a one-step delay. The world is a torus. It is the blockage that makes the problem solvable for Robject. The rows represent different examples and the corresponding best and second-best solutions (see text).

recovers. The six-step solution corresponds to the previously learned case: Robject approaches the prey by taking steps backward.

In the last set of runs, the first experiment was tried again with the trained Robject. Results are shown by the short dashed line with squares (S_3 curve) in Figure 12. In this trial, Robject finds the solution after 40 epochs, and its performance is better than for the untrained case from the very beginning. The found solution, which is composed of five steps, is the best solution. It and a seven-step solution are shown in the bottom line of Figure 11. The seven-step solution is the generalization of the six-step solution of the S_2 problem and, in some cases, Robject uses that solution

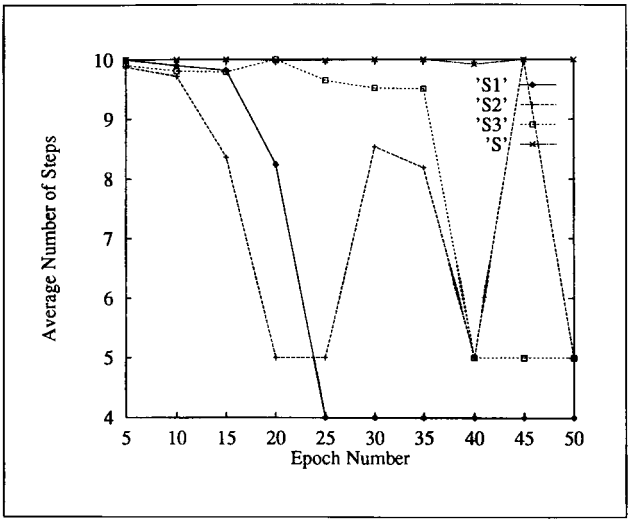


Figure 12
Average number of steps for the nonprobabilistic world. S_1 curve (with solid line and diamonds) represents the experimental results for the top row case of Figure 11. S_2 curve (with the + signs and diamonds) represents the results for the middle row of Figure 11, with Robject pretrained on the S_1 problem. S_3 is similar but is pretrained on S_1 and then S_2 cases. The dotted S curve (with crosses) corresponds to the S_3 case without pretraining.

before it finds the optimal one. The five-step solution is new in one respect: In all the other cases, Robject approached the prey, whereas in this case (we are on a torus where Robject sees the world with itself in the middle), Robject's first move was in the other direction; it moved away from the prey and put the prey in the trap.

Conclusions

The model we are suggesting is based on a brain model (Csányi, 1982). It appears to be one possibility and a firm base for building an adaptive, self-organizing autonomous agent that (1) makes extensive use of ANNs, (2) is parallel at every level, (3) collects experiences, (4) reduces rules from those experiences, (5) may be connected to expert systems, and (6) has efficient options to avoid combinatorial breakdown.

The model we have been investigating is relatively small and has only two goals at present—stop hunger and remove pain. The goal of pain removal is not too complicated and is learned very quickly. Later, the pain buffer efficiently inhibits situations with pain. As a result, the problem of goal conflicts has not been faced in the experiments. This will be different if pain is treated later as a negative goal.

It was shown that the model works in probabilistic and conditionally probabilistic problems, but it is not built on the estimation of probabilities and measures. In our opinion, the most realistic problem contains nonmeasurable subproblems that can limit approaches with, for example, cost functions. It is rather the collected set of experiences that helps us to judge in situations, so the system collects experiences

and then makes a decision with the help of a spreading activation model followed by a competing (winner-take-all) stage that leads to new experiences.

The model may be taught, can be prewired, or may contain adaptive or inherited subsystems. It is robust in the sense that there is a broad range wherein the performance is a shallow function of the parameters.

Generalization capabilities are severely restricted, but the chasing (conditionally probabilistic space) problem shows that the model might have an insight into the problem. It is interesting that all the limitations we see about the model may be solved with neural network solutions; in fact, these fit the model the best. Considering the generalization problem, for example, it seems that feature extraction of the states when reaching a goal could lead to (1) breaking the problem up to subgoals and (2) improving the generalization capabilities through establishing the crucial components of goal fulfillments. In this respect, it was established that the model works better and more efficiently on a family of problems (see, for example, the long-maze problem).

There is another property of organisms that should be built into the systems, this being the drive to practice for more efficient problem solving. The precondition for practicing is the search for reproducing situations—that is, to find routes that lead to cases experienced previously. Thus, there is the need for a special goal, namely the goal of inverting a problem.

Another finding of these computer experiments is that it is not advisable to lengthen the STM beyond limits. If this is done, the model then remembers and also insists on awkward solutions and is therefore not forced to find simple solutions. On the other hand, STM does not allow complicated problems to be solved, which leads to the need for operator (action) fusion (i.e., to the need for combining and memorizing efficient time series of operators as compound operators that could be used as a single action). This, again, could be a subject of feature extraction.

It is important to avoid, or at least to realize the presence of, traps during the course of learning. The model offers a solution to avoid traps: Triplets in the STM could be inhibited in the kernel and cannot be the winner of an upcoming decision.

In conclusion, this model that works with self-organizing cues and competing concepts seems promising and flexible for self-organizing autonomous agents.

Acknowledgments

We are grateful to Dr. Klára Konrád and Dr. Iván Futó for helpful discussions. This work was supported in part by Országos Tudományos Kutatási Alap grant no. 1890/1991 and the U.S.-Hungarian Joint Fund grant no. 91a-168.

References

Agranat, A. J., Neugebauer, C. F., & Yariv, Y. (1990). The CCD neural processor: A neural network integrated circuit with 65536 programmable synapses. *IEEE*

Transactions on Circuits and Systems, 37, 1073–1075.
Agranat, A. J., & Yariv, Y. (1987). Semiparallel microelectronic implementation of neural network models using CCD technology. *Electronics Letters, 23*, 580–582.
Anderson, C. W. (1987). Strategy learning with multilayer connectionist representation. In *Proceedings of the Fourth International Workshop on Machine Learning*, Ann Arbor, MI.
Barto, A., Bradtke, S., & Singh, S. (1991). *Real-time learning and control using asynchronous dynamic programming* (Tech. Rep. No. 91-57). Boston: Computer Science Department, University of Massachusetts.
Bundy, A. (Ed.). (1990). *Catalogue of artificial intelligence techniques*. (3rd ed.). Heidelberg: Springer-Verlag.
Carpenter, G. A., & Grossberg, S. A. (1987). Massively parallel architecture for self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing, 37*, 54–115.
Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Sydney, Australia.
Cloak, F. T., Jr. (1975). Is cultural ecology possible? *Human Ecology, 3*, 161–182.
Collins, A., & Loftus, E. (1975). A spreading activation theory of semantic processing. *Psychological Review, 82*, 407–428.
Csányi, V. (1982). *General theory of evolution*. Budapest: Akadémiai Könyvkiadó.
Földiák, P. (1991). Learning invariance from transformation sequences. *Neural Computation, 3*(2), 194–200.
Fomin, T., & Lőrincz, A. (1993). *On the potential of Hebbian and anti-Hebbian learning*. Manuscript submitted for publication.
Fukushima, K. (1992). Character recognition with neural networks. *Neural Computing, 4*, 221–233.
Grossberg, S. A. (1968). Some nonlinear networks capable of learning a spatial pattern of arbitrary complexity. *Proceedings of the National Academy of Sciences, 59*, 368–372.
Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.
Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the theory of neural computation*. Redwood City: Addison-Wesley.
Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*, 359–366.
Huberman, B. A., & Hogg, T. (1987). Phase transition in artificial intelligence systems. *Artificial Intelligence, 33*, 155–171.
Judd, J. S. (1990). *Neural network design and the complexity of learning*. Cambridge, MA: MIT Press.
Korf, R. E. (1990). Real time heuristic search. *Artificial Intelligence, 42*, 189–211.
Lin, L.-J. (1990). *Self-improving reactive agents: Case studies of reinforcement learning framework*. (Tech. Rep. No. CMU-CS-90-109). Pittsburgh: Carnegie-Mellon University.
Maes, P. (1992). Learning behavior networks from experience. In F. J. Varela & P. Bourgine (Eds.), *Toward a practice of autonomous systems: Proceedings of the First European Conference on Artificial Life*. Cambridge, MA: MIT Press.

Olshausen, B., Anderson, C., & Van Essen, D. (1993). *A neural model of visual attention and invariant pattern recognition.* Manuscript submitted for publication.

Peng, Y., & Reggia, J. A. (1989). A connectionist model for diagnostic problem solving. *IEEE Transactions on Systems, Man and Cybernetics, 19*, 285–298.

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

Szepesvári, C., Balázs, L., & Lőrincz, A. (in press). Topology learning solved by extended objects: A neural network model. *Neural Computation.*

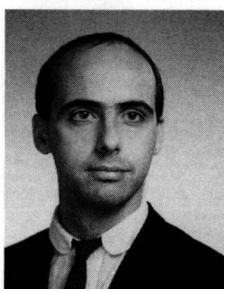
Thagard, P. (1989). Explanatory coherence. *Behavioral and Brain Sciences, 12*, 435–467.

Varela, F. J., & Bourgine, P. (Eds.) (1992). *Toward a practice of autonomous systems: Proceedings of the First European Conference on Artificial Life.* Cambridge, MA: MIT Press.

Watkins, C. J. C. H. (1989). *Learning from delayed reward.* Unpublished doctoral dissertation, Kings College, Cambridge, England.

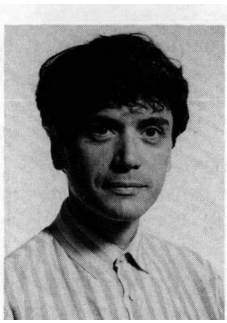
Widrow, B., Gupta, N. K., & Maitra, S. (1973). Punish/reward: Learning with critic in adaptive threshold systems. *IEEE Transactions on Systems, Man and Cybernetics SMC-3*, 455–465.

About the Authors



Csaba Szepesvári

Csaba Szepesvári graduated from the Attila József University of Szeged in 1993. He won the Main Prize of the National Conference of Student Researchers for his Master's thesis on "Role of Local Connections in Competitive Neural Networks: Collective Learning and Representation of Geometry." Currently he is a Ph.D. student at the Attila József University of Szeged working on self-organizing artificial neural networks and autonomous systems.



András Lőrincz

András Lőrincz graduated from the Loránd Eötvös University, Budapest in 1975. He wrote his Master's thesis on the field of theoretical particle physics, titled "Statistical Many Particle Creation." He did experimental Ph.D. work on "Thermocurrent Dosimetry with Al_2O_3 Single Crystals" in 1978, and wrote his Candidate thesis in both experimental and theoretical studies of "Ultrafast Energy Redistribution Processes of Molecules in Supersonic Expansions" in 1986. Currently he is working on control, optimal control, adaptive control and their applications to quantum systems and autonomous systems. He is a member of the Loránd Eötvös Society of Physics.