MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 1180 December 1989

How To Do the Right Thing

Pattie Maes

Abstract

This paper presents a novel approach to the problem of action selection for an autonomous agent. An agent is viewed as a collection of competence modules. Action selection is modeled as an emergent property of an activation/inhibition dynamics among these modules. A concrete action selection algorithm is presented and a detailed account of the results is given. This algorithm combines characteristics of both traditional planners and reactive systems: it produces fast and robust activity in a tight interaction loop with the environment, while at the same time allowing for some prediction and planning to take place. It provides global parameters, which one can use to tune the action selection behavior to the characteristics of the task environment. As such one can smoothly trade off goal-orientedness for situation-orientedness, bias towards ongoing plans (inertia) for adaptivity, thoughtfulness for speed, and adjust its sensitivity to goal conflicts.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Supported by Siemens with additional support from the University Research Initiative under Office of Naval Research contract N00014–86–K–0685, and the Defense Advanced Research Projects Agency under Office of Naval Research contract N00014–85–k–0124.

1 Introduction

This paper addresses the following problem. Imagine an autonomous agent which has to achieve a number of global goals in a complex dynamic environment. An example could be a rover that has to explore Mars and collect samples of soil. How can such an agent select 'the most appropriate' or 'the most relevant' next action to take at a particular moment, when facing a particular situation? Important constraints are that the world is too complex to be entirely predictable and that the agent has limited computational resources and limited time resources. This implies that the action selection cannot be completely 'rational' or optimal. It should, however, be robust, fast, and make 'good enough' decisions (Simon, 1955). By 'good enough' we mean, among other things, that the action selection behavior should demonstrate the following characteristics:

- it favors actions that are goal-oriented, in particular, actions that contribute to several goals at once,
- it favors actions that are relevant to the current situation, in particular it exploits opportunities and is highly adaptive to unpredictable and changing situations,
- it favors actions that contribute to the ongoing goal/plan (unless another action rates a lot better), i.e., it 'sticks' onto a particular goal unless there is a good reason to start working on something different.
- it looks ahead (or 'plans'), in particular to avoid hazardous situations and handle interacting and conflicting goals,
- it is robust (never completely breaks down), even when certain components fail,
- and it is reactive and fast.

The paper studies this problem in the context of the Society of the Mind theory (Minsky, 1986) to which the Subsumption Architecture (Brooks, 1986) is also related. This theory suggests the building of an intelligent system as a society of interacting, mindless agents, each having their own specific competence. For example, a society of agents that is able to build a tower would incorporate 'competence modules' for finding a block, for grasping a block, for moving a block, etc. The idea is that competence modules cooperate (locally) in such a way that the society as a whole functions properly. Such an architecture is very attractive because of its distributedness, modular structure, emergent global functionality and robustness.

One of the open problems is how action can be controlled in such a distributed system. More specifically: (i) how is it determined whether or not some competence module should become active (take some real world actions by controlling the effectors) at a specific moment, and (ii) what are the factors that determine cooperation among certain competence modules. Several solutions can be adopted. One approach is to hand-code (and by that hard-wire) the control flow among the competence modules (Brooks, 1986). Another approach is to introduce a hierarchical structure to tell competence modules whether they are allowed to perform an action or not. This paper investigates yet another, entirely different type of solution.

The hypotheses that are tested are:

- 'good enough' action selection of the global system can be obtained by letting the competence modules activate and inhibit each other in the right way,
- no 'bureaucratic' competence modules are necessary (i.e., modules whose only competence is determining which other modules should be activated or inhibited) nor do we need global forms of control.

We are studying the adequacy of these hypotheses are attempting to determine which activation/inhibition dynamics is appropriate. To this end we are developing a series of algorithms and testing them in computer simulations. One such algorithm was discussed in (Maes, 1989). This paper describes a variation on the algorithm which is simpler and produces more interesting results[1].

Experiments have been performed for several applications. The resulting systems do exhibit the desired properties of goal-orientedness, situation-orientedness, adaptivity, robustness, looking ahead, etc. Further, global parameters make it possible to smoothly mediate between these action selection criteria, such as trading off goal-orientedness for data-orientedness, adaptivity for inertia, sensitivity to goal conflicts and thoughtfulness for speed.

One cannot classify this algorithm as either belonging to the traditional AI approach (in which competence is programmed) or to the connectionist approach (in which competence is the result of tabula rasa learning). Nor is it a hybrid system in the sense that there would be a distinct symbolic and subsymbolic component. Instead, the algorithm completely integrates characteristics of both approaches by using a connectionist computational model on a symbolic, structured representation. By doing so, it combines the best of both worlds:

- From connectionism it inherits the interesting properties of intrinsic parallelism, fault-tolerance, sophisticated retrieval and matching capabilities,

[1] In particular, this algorithm also makes use of 'inhibition' among modules, which makes it possible to deal with interacting goals. Further, there are new results on how the global parameters can be used to tune the action selection behavior along different dimensions.

density (or continuity) and global emergent computation from uniform local interaction rules. On the other hand, it avoids putting the whole burden on learning and classification (without excluding the possibility of applying the learning techniques developed in this area).

- From symbolic AI, it adopts representation and structuring principles. The network is prewired, its links have specific meanings which can be understood (such as causality) and nodes are large, meaningful units. Thus, the algorithm inherits such interesting properties as explanation facilities and programmability (the network can be augmented by hand). It further provides a compositional solution to the problem of action selection, which means that the same parts are reused for different problems (e.g. the same network can be given different goals at different times). As a consequence, the networks are smaller (and therefore might prove to be easier to learn or improve). On the other hand, the algorithm avoids problems of traditional AI solutions such as seriality/slowness, brittleness, rigidity, and the communication complexity of distributed AI systems.

This paper is structured as follows: section 2 introduces the algorithm for action selection, section 3 presents a mathematical model, section 4 sketches how it works, section 5 discusses the empirical results obtained, section 6 reflects on the limits of the current algorithm, section 7 compares the algorithm with related work, and finally, section 8 draws some conclusions.

2 Algorithm

An autonomous agent is viewed as a set of competence modules. These competence modules resemble the operators of a classical planning system. A *competence module* i can be described by a tuple $(c_i, a_i, d_i, \alpha_i)$. c_i is a list of preconditions which have to be fulfilled before the competence module can become active. a_i and d_i represent the expected effects of the competence module's action in terms of an add list and a delete list. In addition, each competence module has a level of activation α_i. A competence module is *executable* at time t when all of its preconditions are observed to be true at time t. An executable competence module whose activation-level surpasses a threshold may be selected, which means that it performs some real world actions. The operation of a competence module (what computation it performs, what actions it takes and how) is not made explicit, i.e., competence modules could be hard-wired inside, they could perform logical inference, or whatever.

Competence modules are linked in a network through three types of links: successor links, predecessor links, and conflicter links. The description of the

competence modules of an autonomous agent in terms of a precondition list, add list and delete list completely defines this network:

- There is a *successor link* from competence module x to competence module y ('x has y as successor') for every proposition p that is a member of the add list of x and also member of the precondition list of y (so more than one successor link between two competence modules may exist). Formally, given competence module $x = (c_x, a_x, d_x, \alpha_x)$ and competence module $y = (c_y, a_y, d_y, \alpha_y)$, there is a successor link from x to y, for every proposition $p \in a_x \cap c_y$.

- A *predecessor link* from module x to module y ('x has y as predecessor') exists for every successor link from y to x. Formally, given competence module $x = (c_x, a_x, d_x, \alpha_x)$ and competence module $y = (c_y, a_y, d_y, \alpha_y)$, there is a predecessor link from x to y, for every proposition $p \in c_x \cap a_y$.

- There is a *conflicter link* from module x to module y ('y conflicts with x') for every proposition p that is a member of the delete list of y and a member of the precondition list of x. Formally, given competence module $x = (c_x, a_x, d_x, \alpha_x)$ and competence module $y = (c_y, a_y, d_y, \alpha_y)$, there is a conflicter link from x to y, for every proposition $p \in c_x \cap d_y$.

The intuitive idea is that modules use these links to activate and inhibit each other, so that after some time the activation energy accumulates in the modules that represent the 'best' actions to take given the current situation and goals. Once the activation level of such a module surpasses a certain threshold, and provided the module is executable, it becomes active and takes some real actions. The pattern of spreading activation among modules, as well as the input of new activation energy into the network is determined by the current state of the environment and the current global goals of the agent:

- *Activation by the State*
  There is input of activation energy coming from the state of the environment towards modules that partially match the current state[2]. A competence module is said to partially match the current state if at least one of its preconditions is observed to be true.

- *Activation by the Goals*
  A second source of activation energy is the global goals of the agent. They

[2]Notice that we do not make the assumption that there is a global continuously updated world model. In a real robot, each proposition would be delivered by a virtual sensor, which is a module that decides upon the basis of real sensor data whether a certain proposition should be considered true.

increase the activation level of modules that achieve one of the global goals. A module is said to achieve one of the global goals if one of the goals is a member of the add list of the competence module. Notice that we distinguish two types of goals: *once-only* goals have to be achieved only once, i.e. as soon as they are achieved, they are deleted from the list of global goals.*Permanent* goals have to be achieved continuously. An example of the first is the goal 'spray-paint-car', an example of the second would be 'battery-50

- *Inhibition by the Protected Goals*
  Further, there is an external inhibition (or removal of activation) by the global goals of the agent that have already been achieved and should be protected. These 'protected goals' remove some of the activation from the modules that would undo them. A module is said to undo one of the protected goals when one of the protected goals is member of the delete list of the module.

These processes are continuous: there is a continual flow of activation energy towards the modules that partially match the current state and towards the modules that realize one of the global goals (at every timestep their activation levels are increased). There is a continual decrease of the activation level of the modules that undo the protected goals. This means that the state of the environment and the global goals may change unpredictably at any moment in time. If this happens, the external input of activation automatically flows to other competence modules.

Besides the impact on activation levels from the state and goals, competence modules also activate and inhibit each other. Modules spread activation along their links as follows:

- *Activation of Successors*
  An executable competence module x spreads activation forward. It increases (by a fraction of its own activation level) the activation level of those successors y for which the shared proposition $p \in a_x \cap c_y$ is not true. Intuitively, we want these successor modules to become more activated because they are 'almost executable', since more of their preconditions will be fulfilled after the competence module has become active. Formally, given that competence module $x = (c_x, a_x, d_x, \alpha_x)$ is executable, it spreads forward through those successor links for which the proposition that defines them $p \in a_x$ is false.

- *Activation of Predecessors*
  A competence module x that is not executable spreads activation backward. It increases (by a fraction of its own activation level) the activation level of those predecessors y for which the shared proposition $p \in c_x \cap a_y$ is not true. Intuitively, a non-executable competence module spreads to the modules that 'promise' to fulfill its preconditions that are not yet true, so that the

competence module may become executable afterwards. Formally, given that competence module $x = (c_x, a_x, d_x, \alpha_x)$ is not executable, it spreads backward through those predecessor links for which the proposition that defined them $p \in c_x$ is false.

- *Inhibition of Conflicters*
  Every competence module x (executable or not) decreases (by a fraction of its own activation level) the activation level of those conflicters y for which the shared proposition $p \in c_x \cap d_y$ is true. Intuitively, a module tries to prevent a module that undoes its true preconditions from becoming active. Notice that we do not allow a module to inhibit itself (while it may activate itself). In case of mutual conflict of modules, only the one with the higher activation level inhibits the other. This prevents the phenomenon that the most relevant modules eliminate each other. Formally, competence module $x = (c_x, a_x, d_x, \alpha_x)$ takes away activation energy through all of its conflicter links for which the proposition that defines them $p \in c_x$ is true, except those links for which there exists an inverse conflicter link that is stronger.

The global algorithm performs a loop, in which at every timestep the following computation takes place over all of the competence modules:

1. The impact of the state, goals and protected goals on the activation level of a module is computed.

2. The way the competence module activates and inhibits related modules through its successor links, predecessor links and conflicter links is computed.

3. A decay function ensures that the overall activation level remains constant.

4. The competence module that fulfills the following three conditions becomes active: (i) It has to be executable, (ii) Its level of activation has to surpass a certain threshold and (iii) It must have a higher activation level than all other competence modules that fulfill conditions (i) and (ii). When two competence modules fulfill these conditions (i.e., they are equally strong), one of them is chosen randomly. The activation level of the module that has become active is reinitialized to 0 [3]. If none of the modules fulfills conditions (i) and (ii), the threshold is lowered by 10%.

These four steps are repeated infinitely. Interesting global observable properties are: the sequence of competence modules that have become active, the optimality of this sequence (which is computed by a domain-dependent function), and the

[3] If this were not the case, modules could become active a couple of times in a row without this really being desirable.

speed with which it was obtained (the number of timesteps a competence module has become active relative to the total number of timesteps the system has been running).

Four *global parameters* can be used to 'tune' the spreading activation dynamics, and thereby the action selection behavior of the agent:

1. θ, the threshold for becoming active, and related to it, π the mean level of activation. θ is lowered with 10% each time none of the modules could be selected. It is reset to its initial value when a module could be selected.

2. ϕ, the amount of activation energy a proposition that is observed to be true injects into the network.

3. γ, the amount of activation energy a goal injects into the network.

4. δ, the amount of activation energy a protected goal takes away from the network.

These parameters also determine the amount of activation that modules spread forward, backward or take away. More precisely, for each false proposition in its precondition list, a non-executable module spreads α to its predecessors. For each false proposition in its add list, an executable module spreads $\alpha\frac{\phi}{\gamma}$ to its successors. For each true proposition in its precondition list a module takes away $\alpha\frac{\delta}{\gamma}$ from its conflictors. These factors were chosen this way because the internal spreading of activation should have the same semantics/effects as the input/output by the state and the goals. The ratios of input from the state versus input from the goals versus output by the protected goals are the same as the ratios of input from predecessors versus input from successors versus output by modules with which a module conflicts. Intuitively, we want to view preconditions that are not yet true as subgoals, effects that are about to be true as 'predictions', and preconditions that are true as protected subgoals.

The algorithm as it is described until now, has a drawback that has to be dealt with. The length of a precondition list, add list or delete list affects the input and output of activation to a module. In particular, a module which has a lot of propositions in its add list and precondition list has more sources of activation energy than a module that only has a few. Therefore, all input of activation to a module or removal of activation from a module is weighted by $\frac{1}{n}$, where n is (i) the number of propositions in the precondition list (in the case of input coming from the state and from the predecessors), (ii) the number of propositions in the add-list (in the case of input from the goals or from successors), or (iii) the number of propositions in the delete list (in the case of removal of activation by the protected goals or by modules with whom the module conflicts).

Finally, we want modules that achieve the same goal or modules that use the same precondition to compete with one another to become active (we view them as representing a disjunction or choice point). Therefore, the amount of activation that is spread or taken away for a particular proposition is split among the affected modules. For example, for a particular proposition p that is observed to be true the state divides ϕ among all of the modules that have that precondition in their precondition list. The same not only holds for the effect of the goals and the protected goals, but also for the internal spreading of activation. For example when a large number of modules achieve a precondition of module m, the activation α_m that m spreads backward for that proposition is equally divided among all of these modules. When on the other hand there is only one other module that can make this precondition true, module m increases the activation level of that module by its own activation level α_m. One implicit assumption on which this is based is that the preconditions are in conjunctive normal form. A disjunction of two preconditions would be represented by a single proposition, for which two competence modules exist that can make it true.

3 Mathematical Model

This section of the paper presents a mathematical description of the algorithm so as to make reproduction of the results possible. Given:

- a set of competence modules $1..n$,
- a set of propositions P,
- a function $S(t)$ returning the propositions that are observed to be true at time t (the state of the environment as perceived by the agent); S being implemented by an independent process (or the real world),
- a function $G(t)$ returning the propositions that are a goal of the agent at time t; G being implemented by an independent process,
- a function $R(t)$ returning the propositions that are a goal of the agent that has already been achieved at time t; R being implemented by an independent process (e.g. some internal or external goal creator),
- a function $executable(i,t)$, which returns 1 if competence module i is executable at time t (i.e., if all of the preconditions of competence module i are members of $S(t)$), and 0 otherwise.
- a function $M(j)$, which returns the set of modules that match proposition j, i.e., the modules x for which $j \in c_x$,

- a function $A(j)$, which returns the set of modules that achieve proposition j, i.e., the modules x for which $j \in a_x$,
- a function $U(j)$, which returns the set of modules that undo proposition j, i.e., the modules x for which $j \in d_x$,
- π, the mean level of activation,
- θ, the threshold of activation, where θ is lowered 10% every time no module could be selected, and is reset to its initial value whenever a module becomes active.
- ϕ, the amount of activation energy injected by the state per true proposition,
- γ, the amount of activation energy injected by the goals per goal,
- δ, the amount of activation energy taken away by the protected goals per protected goal.

Given competence module $x = (c_x, a_x, d_x, \alpha_x)$, the input of activation to module x from the state at time t is:

$$input\_from\_state(x,t) = \sum_j \phi \frac{1}{\#M(j)} \frac{1}{\#c_x}$$

where $j \in S(t) \cap c_x$ and where $\#$ stands for the cardinality of a set.
The input of activation to competence module x from the goals at time t is:

$$input\_from\_goals(x,t) = \sum_j \gamma \frac{1}{\#A(j)} \frac{1}{\#a_x}$$

where $j \in G(t) \cap a_x$.
The removal of activation from competence module x by the goals that are protected at time t is:

$$taken\_away\_by\_protected\_goals(x,t) = \sum_j \delta \frac{1}{\#U(j)} \frac{1}{\#d_x}$$

where $j \in R(t) \cap d_x$.
The following equation specifies what a competence module $x = (c_x, a_x, d_x, \alpha_x)$ spreads backward to a competence module $y = (c_y, a_y, d_y, \alpha_y)$:

$$spreads\_bw(x,y,t) = \begin{cases} \sum_j \alpha_x(t-1) \frac{1}{\#A(j)} \frac{1}{\#a_y} & \text{if } executable(x,t) = 0 \\ 0 & \text{if } executable(x,t) = 1 \end{cases}$$

where $j \notin S(t) \wedge j \in c_x \cap a_y$.

The following equation specifies what module x spreads forward to module y:

$$spreads\_fw(x,y,t) = \begin{cases} \sum_j \alpha_x(t-1)\frac{\phi}{\gamma}\frac{1}{\#M(j)}\frac{1}{\#c_y} & \text{if } executable(x,t)=1 \\ 0 & \text{if } executable(x,t)=0 \end{cases}$$

where $j \notin S(t) \wedge j \in a_x \cap c_y$.
The following equation specifies what module x takes away from module y:

$takes\_away(x,y,t) =$

$$\begin{cases} 0 & \text{if } (\alpha_x(t-1) < \alpha_y(t-1)) \wedge (\exists i \in S(t) \cap c_y \cap d_x) \\ max(\sum_j \alpha_x(t-1)\frac{\delta}{\gamma}\frac{1}{\#U(j)}\frac{1}{\#d_y}, \alpha_y(t-1)) & \text{otherwise} \end{cases}$$

where $j \in c_x \cap d_y \cap S(t)$.
The activation level of a competence module y at time t is defined as:

$$\begin{aligned} \alpha(y,0) &= 0 \\ \alpha(y,t) &= decay(\alpha(y,t-1)(1-active(y,t-1)) \\ &\quad + input\_from\_state(y,t) + input\_from\_goals(y,t) \\ &\quad - taken\_away\_by\_protected\_goals(y,t) \\ &\quad + \sum_{x,z}(spreads\_bw(x,y,t) + spreads\_fw(x,y,t) - takes\_away(z,y,t))) \end{aligned}$$

where x ranges over the modules of the network, z ranges over the modules of the network minus the module y, $t > 0$, and the decay function is such that the global activation remains constant:

$$\sum_y \alpha_y(t) = n\pi$$

The competence module that becomes active at time t is module i such that:

$$active(t,i) = 1 \; if \begin{cases} \alpha(i,t) >= \theta & (1) \\ executable(i,t) = 1 & (2) \\ \forall j \; fulfilling(1) \wedge (2) : \alpha(i,t) >= \alpha(j,t) & (3) \end{cases}$$

$$active(t,i) = 0 \; otherwise$$

4 Example

This section illustrates the algorithm with a concrete, simple example. Later in the paper more interesting examples are discussed. The example is taken from the planning chapter of (Charniak & Mc Dermott, 1985). It involves a robot with two hands which has to spray-paint itself and sand a board. The task has some

complexity to it. The robot has to coordinate the use of its hands or otherwise be clever enough to use a vise to hold the board and perform the jobs in parallel. Furthermore, it should perform the sanding of the board first, because once it has painted itself, it is no longer operational. The definition of the competence modules in terms of their precondition lists, add lists and delete lists is presented in figure 1.

On the basis of these definitions the spreading activation network in figure 2 is constructed. A possible solution to the problem would be to pick up the board, put it in the vise, pick up the sander, sand the board in the vise, pick up the sprayer and spray paint itself.

A (computer-) environment has been built in which the behavior of such a network of competence modules can be simulated. The program is written in Common LISP on a SYMBOLICS machine. Figure 3 shows a bitmap of the system simulating the network described above. The initial state of the environment is *S(0) = (hand-is-empty, hand-is-empty, sander-somewhere, sprayer-somewhere, operational, board-somewhere)*, the initial goals are *G(0) = (board-sanded, self-painted)*.

It is also possible to obtain a trace showing in detail how the spreading activation has evolved. In the remainder of this section, we study the trace of the experiment shown in figure 3 in order to explain its action selection behavior. The activation levels of the competence modules are initialized to zero. At time 1, the modules don't have any activation energy to spread yet, so there is only the input/output from the state and goals. Notice that SAND-BOARD-IN-HAND and SAND-BOARD-IN-VISE have to share the activation energy coming from the goal 'board-sanded'.

```
TIME: 1

state of the environment: (HAND-IS-EMPTY HAND-IS-EMPTY SANDER-SOMEWHERE
                           SPRAYER-SOMEWHERE OPERATIONAL BOARD-SOMEWHERE)
goals of the environment: (BOARD-SANDED SELF-PAINTED)
protected goals of the environment: NIL

state gives PICK-UP-SANDER an extra activation of 3.3333333
state gives PICK-UP-SPRAYER an extra activation of 3.3333333
state gives PICK-UP-BOARD an extra activation of 3.3333333
state gives PICK-UP-SANDER an extra activation of 10.0
state gives PICK-UP-SPRAYER an extra activation of 10.0
state gives SPRAY-PAINT-SELF an extra activation of 3.3333333
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
state gives PICK-UP-BOARD an extra activation of 10.0
goals give SAND-BOARD-IN-HAND an extra activation of 35.0
goals give SAND-BOARD-IN-VISE an extra activation of 35.0
goals give SPRAY-PAINT-SELF an extra activation of 70.0
```

```
(defmodule PICK-UP-SPRAYER
    :condition-list '(sprayer-somewhere hand-is-empty)
    :add-list '(sprayer-in-hand)
    :delete-list '(sprayer-somewhere hand-is-empty))
(defmodule PICK-UP-SANDER
    :condition-list '(sander-somewhere hand-is-empty)
    :add-list '(sander-in-hand)
    :delete-list '(sander-somewhere hand-is-empty))
(defmodule PICK-UP-BOARD
    :condition-list '(board-somewhere hand-is-empty)
    :add-list '(board-in-hand)
    :delete-list '(board-somewhere hand-is-empty))
(defmodule PUT-DOWN-SPRAYER
    :condition-list '(sprayer-in-hand)
    :add-list '(sprayer-somewhere hand-is-empty)
    :delete-list '(sprayer-in-hand))
(defmodule PUT-DOWN-SANDER
    :condition-list '(sander-in-hand)
    :add-list '(sander-somewhere hand-is-empty)
    :delete-list '(sander-in-hand))
(defmodule PUT-DOWN-BOARD
    :condition-list '(board-in-hand)
    :add-list '(board-somewhere hand-is-empty)
    :delete-list '(board-in-hand))
(defmodule SAND-BOARD-IN-HAND
    :condition-list '(operational board-in-hand sander-in-hand)
    :add-list '(board-sanded)
    :delete-list '())
(defmodule SAND-BOARD-IN-VISE
    :condition-list '(operational board-in-vise sander-in-hand)
    :add-list '(board-sanded)
    :delete-list '())
(defmodule SPRAY-PAINT-SELF
    :condition-list '(operational sprayer-in-hand)
    :add-list '(self-painted)
    :delete-list '(operational))
(defmodule PLACE-BOARD-IN-VISE
    :condition-list '(board-in-hand)
    :add-list '(hand-is-empty board-in-vise)
    :delete-list '(board-in-hand))
```

Figure 1: Definition of the competence modules involved in the toy example.

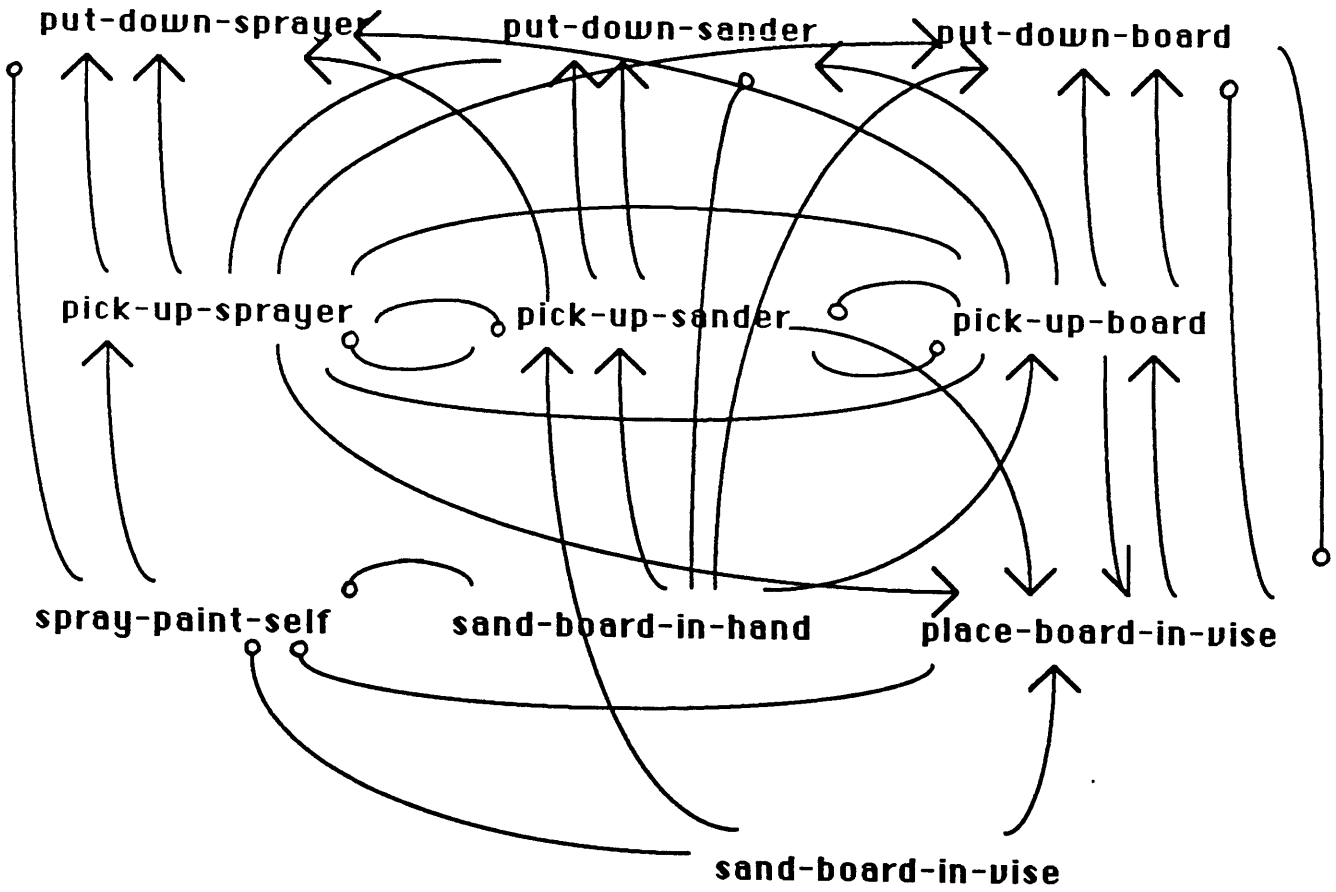

Figure 2: The spreading activation network for the toy example. The predecessor links (from a competence module to its predecessors) are shown as arrows (the symbol of an activation link). The conflicter links are shown as inhibition links (with a little circle at the end). The successor links are not shown (there is a successor link in the inverse direction for every predecessor link).

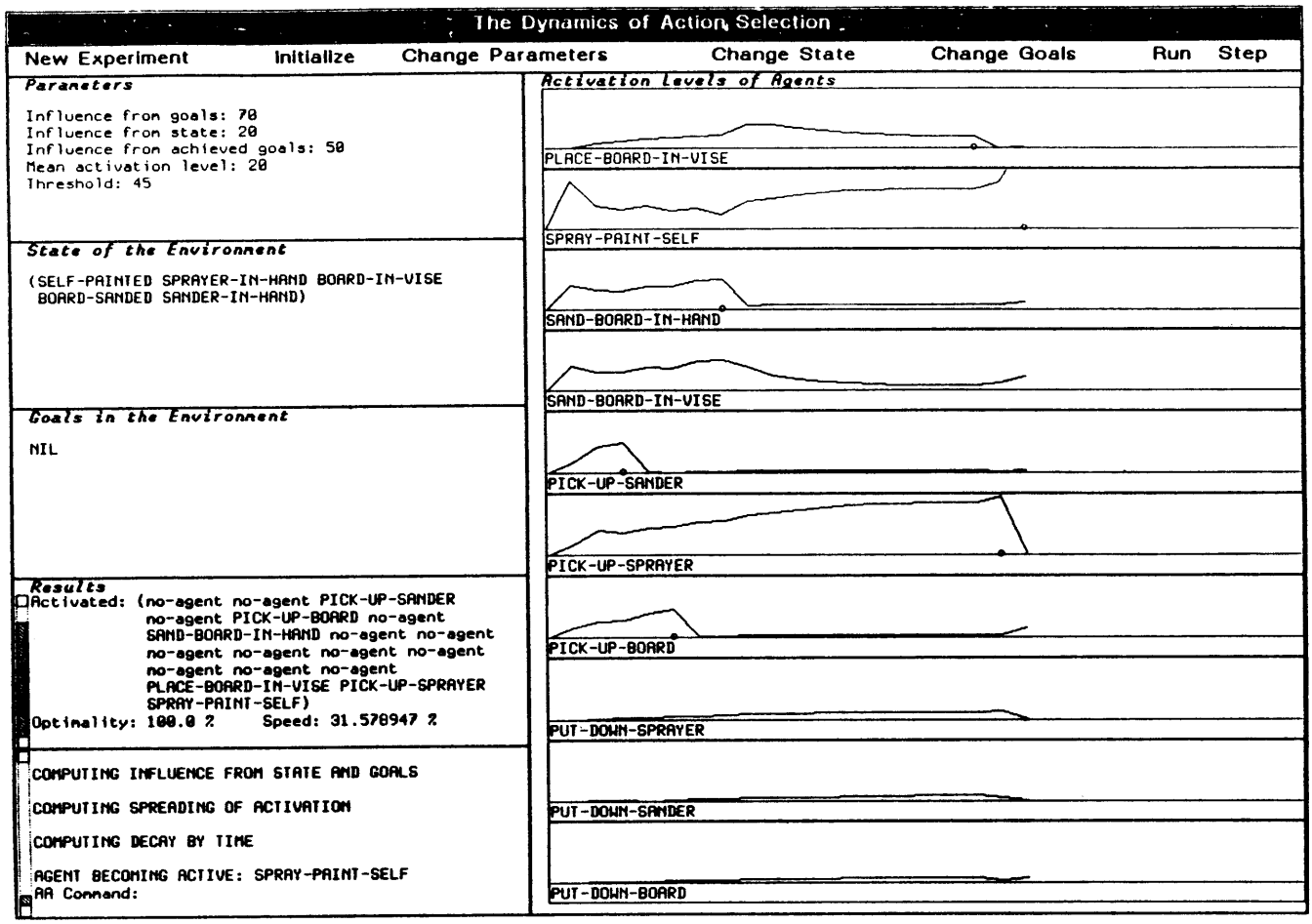

Figure 3: The user interface of the simulation environment. The upper pane is a menu of commands. It makes it possible to define a new network, to initialize the current network, to change the global parameters, to change the state of the environment, to change the goals of the network and to run or step through the behavior of a network. The left-hand panes display the parameters, the current state of the environment, the current goals of the network and the results of the simulation (among which is the list of activated modules). The right-hand panes display the activation levels of competence modules over time (the X-axis represents time, while the Y-axis displays the activation level). The little circles tell when a competence module has become active.

```
activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 0.0
   activation-level SPRAY-PAINT-SELF: 73.333336
   activation-level SAND-BOARD-IN-HAND: 37.22222
   activation-level SAND-BOARD-IN-VISE: 37.22222
   activation-level PICK-UP-SANDER: 13.333333
   activation-level PICK-UP-SPRAYER: 13.333333
   activation-level PICK-UP-BOARD: 13.333333
   activation-level PUT-DOWN-SPRAYER: 0.0
   activation-level PUT-DOWN-SANDER: 0.0
   activation-level PUT-DOWN-BOARD: 0.0

NO MODULE becoming active
threshold is lowered to 40.5
```

None of the executable modules has accumulated enough activation to become active. As a result the threshold is lowered by 10%. At time 2, the input/output from the state and goals is the same as at time 1 (not reprinted). Now there is also some spreading activation among modules. Notice that the modules that match the goals, SPRAY-PAINT-SELF, SAND-BOARD-IN-VISE and SAND-BOARD-IN-HAND spread backwards to their predecessors PICK-UP-SPRAYER, PICK-UP-SAN-DER, PICK-UP-BOARD, and PLACE-BOARD-IN-VISE to make their conditions true. So the false preconditions of the modules that achieve the goals are treated as 'subgoals' by the algorithm.

In case there is only one predecessor for a false precondition, they increase that module's activation level with their own activation level. For example, PICK-UP-SPRAYER receives as much activation as what SPRAY-PAINT-SELF has, because it is the only module that achieves the precondition 'sprayer-in-hand'. Notice further that SAND-BOARD-IN-HAND and SAND-BOARD-IN-VISE weaken SPRAY-PAINT-SELF because it deletes their precondition 'operational'. Finally the executable modules, PICK-UP-SPRAYER, PICK-UP-SANDER and PICK-UP-BOARD activate their successors. This activation is less important than the backward spreading, because we want the impact of goals (and subgoals) to be greater than that of the state (and the 'almost true propositions').

```
TIME: 2

state gives ...

PLACE-BOARD-IN-VISE spreads 0.0 backward to PICK-UP-BOARD for BOARD-IN-HAND
SPRAY-PAINT-SELF spreads 73.333336 backward to PICK-UP-SPRAYER for SPRAYER-IN-HAND
SAND-BOARD-IN-HAND spreads 37.22222 backward to PICK-UP-BOARD for BOARD-IN-HAND
SAND-BOARD-IN-HAND spreads 37.22222 backward to PICK-UP-SANDER for SANDER-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) SPRAY-PAINT-SELF with 26.587301 for OPERATIONAL
SAND-BOARD-IN-VISE spreads 18.61111 backward to PLACE-BOARD-IN-VISE for BOARD-IN-VISE
```

```
SAND-BOARD-IN-VISE spreads 37.22222 backward to PICK-UP-SANDER for SANDER-IN-HAND
SAND-BOARD-IN-VISE decreases (inhibits) SPRAY-PAINT-SELF with 26.587301 for OPERATIONAL
PICK-UP-SANDER spreads 0.42328045 forward to SAND-BOARD-IN-HAND for SANDER-IN-HAND
PICK-UP-SANDER spreads 0.42328045 forward to SAND-BOARD-IN-VISE for SANDER-IN-HAND
PICK-UP-SANDER spreads 1.2698413 forward to PUT-DOWN-SANDER for SANDER-IN-HAND
PICK-UP-SPRAYER spreads 0.95238096 forward to SPRAY-PAINT-SELF for SPRAYER-IN-HAND
PICK-UP-SPRAYER spreads 1.9047619 forward to PUT-DOWN-SPRAYER for SPRAYER-IN-HAND
PICK-UP-BOARD spreads 1.2698413 forward to PLACE-BOARD-IN-VISE for BOARD-IN-HAND
PICK-UP-BOARD spreads 0.42328045 forward to SAND-BOARD-IN-HAND for BOARD-IN-HAND
PICK-UP-BOARD spreads 1.2698413 forward to PUT-DOWN-BOARD for BOARD-IN-HAND
PUT-DOWN-SPRAYER spreads 0.0 backward to PICK-UP-SPRAYER for SPRAYER-IN-HAND
PUT-DOWN-SANDER spreads 0.0 backward to PICK-UP-SANDER for SANDER-IN-HAND
PUT-DOWN-BOARD spreads 0.0 backward to PICK-UP-BOARD for BOARD-IN-HAND

activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 7.447046
   activation-level SPRAY-PAINT-SELF: 35.377182
   activation-level SAND-BOARD-IN-HAND: 28.202648
   activation-level SAND-BOARD-IN-VISE: 28.044096
   activation-level PICK-UP-SANDER: 37.874393
   activation-level PICK-UP-SPRAYER: 37.458195
   activation-level PICK-UP-BOARD: 23.931622
   activation-level PUT-DOWN-SPRAYER: 0.7134894
   activation-level PUT-DOWN-SANDER: 0.4756596
   activation-level PUT-DOWN-BOARD: 0.4756596

NO MODULE becoming active
threshold is lowered to 36.45
```

Again, none of the executable modules is activated enough to be selected. At time 3, the spreading activation patterns remain unchanged, except for the amounts of activation energy that are given or taken away by modules. In particular, PICK-UP-SPRAYER receives less activation from its successor SPRAY-PAINT-SELF, than what PICK-UP-SANDER receives from SAND-BOARD-IN-HAND and SAND-BOARD-IN-VISE together.

```
TIME: 3

state gives ...

PLACE-BOARD-IN-VISE spreads ...

activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 9.699059
   activation-level SPRAY-PAINT-SELF: 29.082869
   activation-level SAND-BOARD-IN-HAND: 27.521559
   activation-level SAND-BOARD-IN-VISE: 27.146523
   activation-level PICK-UP-SANDER: 44.079823
   activation-level PICK-UP-SPRAYER: 32.721424
```

```
   activation-level PICK-UP-BOARD: 24.479343
   activation-level PUT-DOWN-SPRAYER: 2.4768724
   activation-level PUT-DOWN-SANDER: 1.6674367
   activation-level PUT-DOWN-BOARD: 1.1251152

module becoming active: PICK-UP-SANDER
```

The module PICK-UP-SANDER now has accumulated enough activation to become active. As a result the state changes, and thus also the input coming from the state and the internal spreading activation patterns. Notice that SAND-BOARD-IN-VISE and SAND-BOARD-IN-HAND now inhibit PUT-DOWN-SANDER to prevent it from undoing the precondition 'sander-in-hand'. Notice also that PICK-UP-BOARD decreases the activation level of PICK-UP-SPRAYER for the precondition 'hand-is-empty'. This inhibition will become stronger in time because SAND-BOARD-IN-VISE and SAND-BOARD-IN-HAND will be enforced since now more of their preconditions are true.

```
TIME: 4

state of the environment: (SANDER-IN-HAND HAND-IS-EMPTY SPRAYER-SOMEWHERE OPERATIONAL
                          BOARD-SOMEWHERE)
goals of the environment: (BOARD-SANDED SELF-PAINTED)
protected goals of the environment: NIL

state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
state gives PUT-DOWN-SANDER an extra activation of 6.6666665
state gives PICK-UP-SANDER an extra activation of 3.3333333
state gives PICK-UP-SPRAYER an extra activation of 3.3333333
state gives PICK-UP-BOARD an extra activation of 3.3333333
state gives PICK-UP-SPRAYER an extra activation of 10.0
state gives SPRAY-PAINT-SELF an extra activation of 3.3333333
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
state gives PICK-UP-BOARD an extra activation of 10.0
goals give SAND-BOARD-IN-HAND an extra activation of 35.0
goals give SAND-BOARD-IN-VISE an extra activation of 35.0
goals give SPRAY-PAINT-SELF an extra activation of 70.0

PLACE-BOARD-IN-VISE spreads 9.699059 backward to PICK-UP-BOARD for BOARD-IN-HAND
SPRAY-PAINT-SELF spreads 29.082869 backward to PICK-UP-SPRAYER for SPRAYER-IN-HAND
SAND-BOARD-IN-HAND spreads 27.521559 backward to PICK-UP-BOARD for BOARD-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) PUT-DOWN-SANDER with 19.658257 for SANDER-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) SPRAY-PAINT-SELF with 19.658257 for OPERATIONAL
SAND-BOARD-IN-VISE spreads 13.573261 backward to PLACE-BOARD-IN-VISE for BOARD-IN-VISE
SAND-BOARD-IN-VISE decreases (inhibits) PUT-DOWN-SANDER with 19.390373 for SANDER-IN-HAND
SAND-BOARD-IN-VISE decreases (inhibits) SPRAY-PAINT-SELF with 19.390373 for OPERATIONAL
PICK-UP-SANDER spreads 0.0 backward to PUT-DOWN-SANDER for SANDER-SOMEWHERE
PICK-UP-SPRAYER spreads 2.3372447 forward to SPRAY-PAINT-SELF for SPRAYER-IN-HAND
```

```
PICK-UP-SPRAYER spreads 4.6744895 forward to PUT-DOWN-SPRAYER for SPRAYER-IN-HAND
PICK-UP-SPRAYER decreases (inhibits) PICK-UP-SANDER with 5.8431115 for HAND-IS-EMPTY
PICK-UP-SPRAYER decreases (inhibits) PICK-UP-BOARD with 5.8431115 for HAND-IS-EMPTY
PICK-UP-BOARD spreads 2.3313663 forward to PLACE-BOARD-IN-VISE for BOARD-IN-HAND
PICK-UP-BOARD spreads 0.7771221 forward to SAND-BOARD-IN-HAND for BOARD-IN-HAND
PICK-UP-BOARD spreads 2.3313663 forward to PUT-DOWN-BOARD for BOARD-IN-HAND
PICK-UP-BOARD decreases (inhibits) PICK-UP-SANDER with 4.3713117 for HAND-IS-EMPTY
PUT-DOWN-SPRAYER spreads 2.4768724 backward to PICK-UP-SPRAYER for SPRAYER-IN-HAND
PUT-DOWN-SANDER spreads 0.23820525 forward to PICK-UP-SANDER for SANDER-SOMEWHERE
PUT-DOWN-BOARD spreads 1.1251152 backward to PICK-UP-BOARD for BOARD-IN-HAND

activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 13.320736
   activation-level SPRAY-PAINT-SELF: 34.184002
   activation-level SAND-BOARD-IN-HAND: 35.24447
   activation-level SAND-BOARD-IN-VISE: 34.64504
   activation-level PICK-UP-SANDER: 0.12393018
   activation-level PICK-UP-SPRAYER: 40.380215
   activation-level PICK-UP-BOARD: 36.582684
   activation-level PUT-DOWN-SPRAYER: 3.720613
   activation-level PUT-DOWN-SANDER: 0.0
   activation-level PUT-DOWN-BOARD: 1.798291

NO MODULE becoming active
threshold is lowered to 40.5
```

At time 5, the spreading activation pattern is similar to that of time 4. The state and the goals spread activation to the same modules. Also modules keep spreading activation to the same modules, except that now the amounts they give and take away have changed (because the activation levels of the modules at time 4 are different from those at time 3).

```
TIME: 5

state gives ...

PLACE-BOARD-IN-VISE spreads ...

activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 15.370311
   activation-level SPRAY-PAINT-SELF: 27.239319
   activation-level SAND-BOARD-IN-HAND: 34.161552
   activation-level SAND-BOARD-IN-VISE: 33.368526
   activation-level PICK-UP-SANDER: 0.0
   activation-level PICK-UP-SPRAYER: 41.26312
   activation-level PICK-UP-BOARD: 41.91644
   activation-level PUT-DOWN-SPRAYER: 4.2737665
   activation-level PUT-DOWN-SANDER: 0.027907925
   activation-level PUT-DOWN-BOARD: 2.379075
```

```
module becoming active: PICK-UP-BOARD
```

The module that becomes active is PICK-UP-BOARD. The state of the environment changes by the actions performed by this module, so that the input from the state and the internal spreading activation patterns are different at time 6.

```
TIME: 6

state of the environment: (BOARD-IN-HAND SANDER-IN-HAND SPRAYER-SOMEWHERE OPERATIONAL)
goals of the environment: (BOARD-SANDED SELF-PAINTED)
protected goals of the environment: NIL

state gives PLACE-BOARD-IN-VISE an extra activation of 6.6666665
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives PUT-DOWN-BOARD an extra activation of 6.6666665
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
state gives PUT-DOWN-SANDER an extra activation of 6.6666665
state gives PICK-UP-SPRAYER an extra activation of 10.0
state gives SPRAY-PAINT-SELF an extra activation of 3.3333333
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
goals give SAND-BOARD-IN-HAND an extra activation of 35.0
goals give SAND-BOARD-IN-VISE an extra activation of 35.0
goals give SPRAY-PAINT-SELF an extra activation of 70.0

PLACE-BOARD-IN-VISE spreads 0.7319196 forward to PICK-UP-SANDER for HAND-IS-EMPTY
PLACE-BOARD-IN-VISE spreads 0.7319196 forward to PICK-UP-SPRAYER for HAND-IS-EMPTY
PLACE-BOARD-IN-VISE spreads 0.7319196 forward to PICK-UP-BOARD for HAND-IS-EMPTY
PLACE-BOARD-IN-VISE spreads 1.4638392 forward to SAND-BOARD-IN-VISE for BOARD-IN-VISE
PLACE-BOARD-IN-VISE decreases (inhibits) PUT-DOWN-BOARD with 10.978794 for BOARD-IN-HAND
SPRAY-PAINT-SELF spreads 27.239319 backward to PICK-UP-SPRAYER for SPRAYER-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) PLACE-BOARD-IN-VISE with 12.200555 for BOARD-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) PUT-DOWN-BOARD with 12.200555 for BOARD-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) PUT-DOWN-SANDER with 24.40111 for SANDER-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) SPRAY-PAINT-SELF with 24.40111 for OPERATIONAL
SAND-BOARD-IN-VISE spreads 16.684263 backward to PLACE-BOARD-IN-VISE for BOARD-IN-VISE
SAND-BOARD-IN-VISE decreases (inhibits) PUT-DOWN-SANDER with 23.834661 for SANDER-IN-HAND
SAND-BOARD-IN-VISE decreases (inhibits) SPRAY-PAINT-SELF with 23.834661 for OPERATIONAL
PICK-UP-SANDER spreads 0.0 backward to PUT-DOWN-SANDER for SANDER-SOMEWHERE
PICK-UP-SANDER spreads 0.0 backward to PLACE-BOARD-IN-VISE for HAND-IS-EMPTY
PICK-UP-SANDER spreads 0.0 backward to PUT-DOWN-SPRAYER for HAND-IS-EMPTY
PICK-UP-SANDER spreads 0.0 backward to PUT-DOWN-SANDER for HAND-IS-EMPTY
PICK-UP-SANDER spreads 0.0 backward to PUT-DOWN-BOARD for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 5.15789 backward to PLACE-BOARD-IN-VISE for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 5.15789 backward to PUT-DOWN-SPRAYER for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 5.15789 backward to PUT-DOWN-SANDER for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 5.15789 backward to PUT-DOWN-BOARD for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.0 backward to PUT-DOWN-BOARD for BOARD-SOMEWHERE
```

```
PICK-UP-BOARD spreads 0.0 backward to PLACE-BOARD-IN-VISE for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.0 backward to PUT-DOWN-SPRAYER for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.0 backward to PUT-DOWN-SANDER for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.0 backward to PUT-DOWN-BOARD for HAND-IS-EMPTY
PUT-DOWN-SPRAYER spreads 4.2737665 backward to PICK-UP-SPRAYER for SPRAYER-IN-HAND
PUT-DOWN-SANDER spreads 0.0039868467 forward to PICK-UP-SANDER for SANDER-SOMEWHERE
PUT-DOWN-SANDER spreads 0.0013289489 forward to PICK-UP-SANDER for HAND-IS-EMPTY
PUT-DOWN-SANDER spreads 0.0013289489 forward to PICK-UP-SPRAYER for HAND-IS-EMPTY
PUT-DOWN-SANDER spreads 0.0013289489 forward to PICK-UP-BOARD for HAND-IS-EMPTY
PUT-DOWN-BOARD spreads 0.3398679 forward to PICK-UP-BOARD for BOARD-SOMEWHERE
PUT-DOWN-BOARD spreads 0.1132893 forward to PICK-UP-SANDER for HAND-IS-EMPTY
PUT-DOWN-BOARD spreads 0.1132893 forward to PICK-UP-SPRAYER for HAND-IS-EMPTY
PUT-DOWN-BOARD spreads 0.1132893 forward to PICK-UP-BOARD for HAND-IS-EMPTY

activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 18.660385
   activation-level SPRAY-PAINT-SELF: 30.829237
   activation-level SAND-BOARD-IN-HAND: 44.666897
   activation-level SAND-BOARD-IN-VISE: 43.753033
   activation-level PICK-UP-SANDER: 0.50100476
   activation-level PICK-UP-SPRAYER: 49.25829
   activation-level PICK-UP-BOARD: 0.6988567
   activation-level PUT-DOWN-SPRAYER: 5.5557523
   activation-level PUT-DOWN-SANDER: 3.0382743
   activation-level PUT-DOWN-BOARD: 3.0382743

NO MODULE becoming active
threshold is lowered to 40.5
```

Again the spreading activation patterns at time 7 are like those at time 6. In particular SAND-BOARD-IN-HAND will now have received enough activation from the state and the goals to become active. Notice that although PICK-UP-SPRAYER has a very high activation level, it does not become active because not all of its preconditions are fulfilled.

```
TIME: 7

state gives ...

PLACE-BOARD-IN-VISE spreads ...

activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 19.967524
   activation-level SPRAY-PAINT-SELF: 21.800142
   activation-level SAND-BOARD-IN-HAND: 45.89835
   activation-level SAND-BOARD-IN-VISE: 45.175903
   activation-level PICK-UP-SANDER: 1.1233512
   activation-level PICK-UP-SPRAYER: 51.47401
   activation-level PICK-UP-BOARD: 1.2285371
```

```
   activation-level PUT-DOWN-SPRAYER: 6.3068533
   activation-level PUT-DOWN-SANDER: 3.486372
   activation-level PUT-DOWN-BOARD: 3.5389647

module becoming active: SAND-BOARD-IN-HAND
```

As a consequence the state and goals change. The only remaining goal to be achieved is 'self-painted'. In order to do so, the robot has to free at least one hand. Notice that PICK-UP-SPRAYER spreads backwards to the modules that can achieve this, i.e., PLACE-BOARD-IN-VISE, PUT-DOWN-SANDER and PUT-DOWN-BOARD.

```
TIME: 8

state of the environment: (BOARD-SANDED BOARD-IN-HAND SANDER-IN-HAND SPRAYER-SOMEWHERE
                          OPERATIONAL)
goals of the environment: (SELF-PAINTED)
protected goals of the environment: (BOARD-SANDED)

state gives PLACE-BOARD-IN-VISE an extra activation of 6.6666665
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives PUT-DOWN-BOARD an extra activation of 6.6666665
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
state gives PUT-DOWN-SANDER an extra activation of 6.6666665
state gives PICK-UP-SPRAYER an extra activation of 10.0
state gives SPRAY-PAINT-SELF an extra activation of 3.3333333
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
goals give SPRAY-PAINT-SELF an extra activation of 70.0

PLACE-BOARD-IN-VISE spreads 0.9508345 forward to PICK-UP-SANDER for HAND-IS-EMPTY
PLACE-BOARD-IN-VISE spreads 0.9508345 forward to PICK-UP-SPRAYER for HAND-IS-EMPTY
PLACE-BOARD-IN-VISE spreads 0.9508345 forward to PICK-UP-BOARD for HAND-IS-EMPTY
PLACE-BOARD-IN-VISE spreads 1.901669 forward to SAND-BOARD-IN-VISE for BOARD-IN-VISE
PLACE-BOARD-IN-VISE decreases (inhibits) PUT-DOWN-BOARD with 14.262517 for BOARD-IN-HAND
SPRAY-PAINT-SELF spreads 21.800142 backward to PICK-UP-SPRAYER for SPRAYER-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) PLACE-BOARD-IN-VISE with 0.0 for BOARD-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) PUT-DOWN-BOARD with 0.0 for BOARD-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) PUT-DOWN-SANDER with 0.0 for SANDER-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) SPRAY-PAINT-SELF with 0.0 for OPERATIONAL
SAND-BOARD-IN-VISE spreads 22.587952 backward to PLACE-BOARD-IN-VISE for BOARD-IN-VISE
SAND-BOARD-IN-VISE decreases (inhibits) PUT-DOWN-SANDER with 32.2685 for SANDER-IN-HAND
SAND-BOARD-IN-VISE decreases (inhibits) SPRAY-PAINT-SELF with 32.2685 for OPERATIONAL
PICK-UP-SANDER spreads 0.5616756 backward to PUT-DOWN-SANDER for SANDER-SOMEWHERE
PICK-UP-SANDER spreads 0.1404189 backward to PLACE-BOARD-IN-VISE for HAND-IS-EMPTY
PICK-UP-SANDER spreads 0.1404189 backward to PUT-DOWN-SPRAYER for HAND-IS-EMPTY
PICK-UP-SANDER spreads 0.1404189 backward to PUT-DOWN-SANDER for HAND-IS-EMPTY
PICK-UP-SANDER spreads 0.1404189 backward to PUT-DOWN-BOARD for HAND-IS-EMPTY
```

```
PICK-UP-SPRAYER spreads 6.4342513 backward to PLACE-BOARD-IN-VISE for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 6.4342513 backward to PUT-DOWN-SPRAYER for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 6.4342513 backward to PUT-DOWN-SANDER for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 6.4342513 backward to PUT-DOWN-BOARD for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.61426854 backward to PUT-DOWN-BOARD for BOARD-SOMEWHERE
PICK-UP-BOARD spreads 0.15356714 backward to PLACE-BOARD-IN-VISE for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.15356714 backward to PUT-DOWN-SPRAYER for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.15356714 backward to PUT-DOWN-SANDER for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.15356714 backward to PUT-DOWN-BOARD for HAND-IS-EMPTY
PUT-DOWN-SPRAYER spreads 6.3068533 backward to PICK-UP-SPRAYER for SPRAYER-IN-HAND
PUT-DOWN-SANDER spreads 0.49805316 forward to PICK-UP-SANDER for SANDER-SOMEWHERE
PUT-DOWN-SANDER spreads 0.16601773 forward to PICK-UP-SANDER for HAND-IS-EMPTY
PUT-DOWN-SANDER spreads 0.16601773 forward to PICK-UP-SPRAYER for HAND-IS-EMPTY
PUT-DOWN-SANDER spreads 0.16601773 forward to PICK-UP-BOARD for HAND-IS-EMPTY
PUT-DOWN-BOARD spreads 0.5055664 forward to PICK-UP-BOARD for BOARD-SOMEWHERE
PUT-DOWN-BOARD spreads 0.16852213 forward to PICK-UP-SANDER for HAND-IS-EMPTY
PUT-DOWN-BOARD spreads 0.16852213 forward to PICK-UP-SPRAYER for HAND-IS-EMPTY
PUT-DOWN-BOARD spreads 0.16852213 forward to PICK-UP-BOARD for HAND-IS-EMPTY

activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 37.119087
   activation-level SPRAY-PAINT-SELF: 41.70643
   activation-level SAND-BOARD-IN-HAND: 4.422858
   activation-level SAND-BOARD-IN-VISE: 34.181183
   activation-level PICK-UP-SANDER: 1.9284406
   activation-level PICK-UP-SPRAYER: 60.28337
   activation-level PICK-UP-BOARD: 2.0032084
   activation-level PUT-DOWN-SPRAYER: 8.647854
   activation-level PUT-DOWN-SANDER: 4.8363376
   activation-level PUT-DOWN-BOARD: 4.8712296

NO MODULE becoming active
threshold is lowered to 40.5
```

At time 9 till 17 the activation patterns remain the same. SPRAY-PAINT-SELF accumulates activation coming from the goals and spreads this activation further towards its only predecessor, namely PICK-UP-SPRAYER. PICK-UP-SPRAYER spreads the received activation further backwards towards the modules that can make its precondition 'hand-is-empty' true. Because there are many such modules, it takes some time before one of them is selected.

```
TIME: 17

state gives ...

PLACE-BOARD-IN-VISE spreads ...

activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 17.6625
```

```
   activation-level SPRAY-PAINT-SELF: 61.41764
   activation-level SAND-BOARD-IN-HAND: 6.4295135
   activation-level SAND-BOARD-IN-VISE: 6.108067
   activation-level PICK-UP-SANDER: 2.5221777
   activation-level PICK-UP-SPRAYER: 79.323494
   activation-level PICK-UP-BOARD: 2.2743216
   activation-level PUT-DOWN-SPRAYER: 10.060002
   activation-level PUT-DOWN-SANDER: 8.496746
   activation-level PUT-DOWN-BOARD: 5.7055316

module becoming active: PLACE-BOARD-IN-VISE
```

Finally PLACE-BOARD-IN-VISE becomes active, and makes one hand free. As a result PICK-UP-SPRAYER (which had already accumulated enough activation) is executable.

```
TIME: 18

state of the environment: (HAND-IS-EMPTY BOARD-IN-VISE BOARD-SANDED SANDER-IN-HAND
                           SPRAYER-SOMEWHERE OPERATIONAL)
goals of the environment: (SELF-PAINTED)
protected goals of the environment: (BOARD-SANDED)

state gives PICK-UP-SANDER an extra activation of 3.3333333
state gives PICK-UP-SPRAYER an extra activation of 3.3333333
state gives PICK-UP-BOARD an extra activation of 3.3333333
state gives SAND-BOARD-IN-VISE an extra activation of 6.6666665
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
state gives PUT-DOWN-SANDER an extra activation of 6.6666665
state gives PICK-UP-SPRAYER an extra activation of 10.0
state gives SPRAY-PAINT-SELF an extra activation of 3.3333333
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
goals give SPRAY-PAINT-SELF an extra activation of 70.0

PLACE-BOARD-IN-VISE spreads 0.0 backward to PICK-UP-BOARD for BOARD-IN-HAND
SPRAY-PAINT-SELF spreads 61.41764 backward to PICK-UP-SPRAYER for SPRAYER-IN-HAND
SAND-BOARD-IN-HAND spreads 6.4295135 backward to PICK-UP-BOARD for BOARD-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) PUT-DOWN-SANDER with 4.5925097 for SANDER-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) SPRAY-PAINT-SELF with 4.5925097 for OPERATIONAL
SAND-BOARD-IN-VISE decreases (inhibits) PUT-DOWN-SANDER with 4.362905 for SANDER-IN-HAND
SAND-BOARD-IN-VISE decreases (inhibits) SPRAY-PAINT-SELF with 4.362905 for OPERATIONAL
PICK-UP-SANDER spreads 1.2610888 backward to PUT-DOWN-SANDER for SANDER-SOMEWHERE
PICK-UP-SANDER decreases (inhibits) PICK-UP-BOARD with 0.45038888 for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 5.665964 forward to SPRAY-PAINT-SELF for SPRAYER-IN-HAND
PICK-UP-SPRAYER spreads 11.331928 forward to PUT-DOWN-SPRAYER for SPRAYER-IN-HAND
PICK-UP-SPRAYER decreases (inhibits) PICK-UP-SANDER with 14.16491 for HAND-IS-EMPTY
PICK-UP-SPRAYER decreases (inhibits) PICK-UP-BOARD with 14.16491 for HAND-IS-EMPTY
PICK-UP-BOARD spreads 1.1371608 backward to PUT-DOWN-BOARD for BOARD-SOMEWHERE
```

```
PUT-DOWN-SPRAYER spreads 10.060002 backward to PICK-UP-SPRAYER for SPRAYER-IN-HAND
PUT-DOWN-SANDER spreads 1.2138209 forward to PICK-UP-SANDER for SANDER-SOMEWHERE
PUT-DOWN-BOARD spreads 5.7055316 backward to PICK-UP-BOARD for BOARD-IN-HAND

activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 0.0
   activation-level SPRAY-PAINT-SELF: 71.77567
   activation-level SAND-BOARD-IN-HAND: 5.936989
   activation-level SAND-BOARD-IN-VISE: 9.401367
   activation-level PICK-UP-SANDER: 0.6627248
   activation-level PICK-UP-SPRAYER: 89.61452
   activation-level PICK-UP-BOARD: 3.1151197
   activation-level PUT-DOWN-SPRAYER: 11.679616
   activation-level PUT-DOWN-SANDER: 4.077989
   activation-level PUT-DOWN-BOARD: 3.7359893

module becoming active: PICK-UP-SPRAYER
```

And finally, the module SPRAY-PAINT-SELF (which also already had accumulated enough activation) becomes executable and is selected.

```
TIME: 19

state of the environment: (SPRAYER-IN-HAND BOARD-IN-VISE BOARD-SANDED SANDER-IN-HAND OPERATIONA
goals of the environment: (SELF-PAINTED)
protected goals of the environment: (BOARD-SANDED)

state gives SPRAY-PAINT-SELF an extra activation of 5.0
state gives PUT-DOWN-SPRAYER an extra activation of 10.0
state gives SAND-BOARD-IN-VISE an extra activation of 6.6666665
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
state gives PUT-DOWN-SANDER an extra activation of 6.6666665
state gives SPRAY-PAINT-SELF an extra activation of 3.3333333
state gives SAND-BOARD-IN-HAND an extra activation of 2.2222223
state gives SAND-BOARD-IN-VISE an extra activation of 2.2222223
goals give SPRAY-PAINT-SELF an extra activation of 70.0

PLACE-BOARD-IN-VISE spreads 0.0 backward to PICK-UP-BOARD for BOARD-IN-HAND
SPRAY-PAINT-SELF decreases (inhibits) PUT-DOWN-SPRAYER with 51.268337 for SPRAYER-IN-HAND
SAND-BOARD-IN-HAND spreads 5.936989 backward to PICK-UP-BOARD for BOARD-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) PUT-DOWN-SANDER with 4.2407064 for SANDER-IN-HAND
SAND-BOARD-IN-HAND decreases (inhibits) SPRAY-PAINT-SELF with 4.2407064 for OPERATIONAL
SAND-BOARD-IN-VISE decreases (inhibits) PUT-DOWN-SANDER with 6.7152624 for SANDER-IN-HAND
SAND-BOARD-IN-VISE decreases (inhibits) SPRAY-PAINT-SELF with 6.7152624 for OPERATIONAL
PICK-UP-SANDER spreads 0.3313624 backward to PUT-DOWN-SANDER for SANDER-SOMEWHERE
PICK-UP-SANDER spreads 0.0828406 backward to PLACE-BOARD-IN-VISE for HAND-IS-EMPTY
PICK-UP-SANDER spreads 0.0828406 backward to PUT-DOWN-SPRAYER for HAND-IS-EMPTY
PICK-UP-SANDER spreads 0.0828406 backward to PUT-DOWN-SANDER for HAND-IS-EMPTY
PICK-UP-SANDER spreads 0.0828406 backward to PUT-DOWN-BOARD for HAND-IS-EMPTY
```

```
PICK-UP-SPRAYER spreads 0.0 backward to PUT-DOWN-SPRAYER for SPRAYER-SOMEWHERE
PICK-UP-SPRAYER spreads 0.0 backward to PLACE-BOARD-IN-VISE for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 0.0 backward to PUT-DOWN-SPRAYER for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 0.0 backward to PUT-DOWN-SANDER for HAND-IS-EMPTY
PICK-UP-SPRAYER spreads 0.0 backward to PUT-DOWN-BOARD for HAND-IS-EMPTY
PICK-UP-BOARD spreads 1.5575598 backward to PUT-DOWN-BOARD for BOARD-SOMEWHERE
PICK-UP-BOARD spreads 0.38938996 backward to PLACE-BOARD-IN-VISE for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.38938996 backward to PUT-DOWN-SPRAYER for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.38938996 backward to PUT-DOWN-SANDER for HAND-IS-EMPTY
PICK-UP-BOARD spreads 0.38938996 backward to PUT-DOWN-BOARD for HAND-IS-EMPTY
PUT-DOWN-SPRAYER spreads 1.6685166 forward to PICK-UP-SPRAYER for SPRAYER-SOMEWHERE
PUT-DOWN-SPRAYER spreads 0.5561722 forward to PICK-UP-SANDER for HAND-IS-EMPTY
PUT-DOWN-SPRAYER spreads 0.5561722 forward to PICK-UP-SPRAYER for HAND-IS-EMPTY
PUT-DOWN-SPRAYER spreads 0.5561722 forward to PICK-UP-BOARD for HAND-IS-EMPTY
PUT-DOWN-SANDER spreads 0.5825699 forward to PICK-UP-SANDER for SANDER-SOMEWHERE
PUT-DOWN-SANDER spreads 0.19418997 forward to PICK-UP-SANDER for HAND-IS-EMPTY
PUT-DOWN-SANDER spreads 0.19418997 forward to PICK-UP-SPRAYER for HAND-IS-EMPTY
PUT-DOWN-SANDER spreads 0.19418997 forward to PICK-UP-BOARD for HAND-IS-EMPTY
PUT-DOWN-BOARD spreads 3.7359893 backward to PICK-UP-BOARD for BOARD-IN-HAND

activation-levels of modules after decay:
   activation-level PLACE-BOARD-IN-VISE: 0.47223055
   activation-level SPRAY-PAINT-SELF: 139.15305
   activation-level SAND-BOARD-IN-HAND: 10.3814335
   activation-level SAND-BOARD-IN-VISE: 20.512478
   activation-level PICK-UP-SANDER: 1.995657
   activation-level PICK-UP-SPRAYER: 2.4188788
   activation-level PICK-UP-BOARD: 13.538461
   activation-level PUT-DOWN-SPRAYER: 0.47223055
   activation-level PUT-DOWN-SANDER: 0.8035929
   activation-level PUT-DOWN-BOARD: 5.76578

module becoming active: SPRAY-PAINT-SELF
```

5 Results

The algorithm presented in this paper can be modeled by a system of differential equations. This system is however too complicated to solve, so that exact predictions about the resulting action selection behavior are not possible. Nevertheless. important qualitative results can be obtained, for example on possible phase transitions with the growth of parameters, such as the size of the network, the mean fanout of a node, etc (Huberman & Hogg, 1987). We have evaluated the algorithm empirically by performing a wide series of experiments using several example applications. The networks had such diverse properties as being very 'wide', very 'long', containing cycles, local high concentrations of links, unlinked subnetworks, destructive modules, conflicting and mutually conflicting modules, etc. All of the

problems presented were solved for large ranges of parameters.

The simulated societies cannot be said to show a 'jump-first think-never' behavior. They do exhibit planning capabilities. They 'consider' to some extent the effects of a sequence of actions before actually embarking on its execution. If a sequence of competence modules exists that transforms the current situation into the goal state, then this sequence becomes highly activated through the cumulative effect of the forward spreading (starting from the current state) and the backward spreading (starting from the goals). If this sequence potentially implies negative effects, it is weakened by the inhibition rules.

More specifically, goal-relevance of the selected action is obtained through the input from the goals and the backward spreading of activation. Situation relevance and opportunistic behavior are obtained through the input of the state and the spreading of activation forward. Conflicting and interacting goals are taken into account through inhibition by the protected goals and inhibition among conflicting modules. Further, local maxima in the action selection are avoided, provided that the spreading of activation can go on long enough (the threshold is high enough), so that the network can evolve towards the optimal activity pattern. And finally, the algorithm automatically biases towards ongoing plans, because these tend to have a shorter distance between state and goals and are favored by the remains of the past spreading activation patterns. Moreover, the global parameters serve as controls by which one can mediate smoothly among these different action selection characteristics.

The notion of a plan is here very different from the classical one existing in AI. A network does not construct an explicit representation of a single plan, but instead expresses its 'intention' or 'urge' to take certain actions by high activation levels of the corresponding modules. Another important difference is that there is no centralized preprogrammed search process. Instead, the operators (competence modules) themselves select the sequence of operators that are activated, and this in a non-hierarchical, highly distributed way. There is no search tree constructed, i.e., there is no explicit representation built of state changes after taking certain actions.

Consequently, the system does not suffer from the disadvantages of search trees such as: that information is duplicated in several parts of a tree; trees grow exponentially with the size of the problem; trees only make a strict representation of plans possible (impossible to work with uncertainties); etc. In addition, the spreading activation process is a much cheaper operation. Of course these advantages are not cost-free. The action selection produced is less 'rational' than that of the sophisticated deliberative planners built in AI. On the other hand the latter systems, when applied in autonomous agents, suffer from brittleness and slowness. What is particularly interesting about the algorithm presented here is that it provides parameters to mediate between adaptivity, speed and reactivity on the one

hand and thoughtfulness and rationality on the other hand.

The following subsections discuss the results observed in detail.

5.1 Goal-Orientedness

The algorithm selects actions that contribute to the global goals of the agent. Given that g is a global goal of the network, then γ of new activation energy is put into the modules that achieve this goal. These modules will in turn per subgoal (false precondition) increase the activation level of the modules that make this subgoal true, and so on. This backward spreading of activation takes care that modules that contribute to goal g are more activated than modules that don't. Furthermore modules that contribute to different goals (or subgoals) receive activation for each of these goals and will therefore be favored over modules that only contribute to one.

If the agent has more than one goal, modules that contribute to the goal that is 'closest' are favored. 'Closest' here means that the path from the goal-achieving modules to the state-matching modules is the shortest. The algorithm also favors modules that have little competition. For example, if the agent has two goals $g1$ and $g2$ and if there is one module that achieves $g1$ and there are two modules that achieve $g2$ then the algorithm favors the module that achieves $g1$, and therefore the probability of $g1$ being realized first is higher. All of these comments hold for subgoals as well as for goals, since subgoals (false preconditions of modules) are treated the same way as goals.

The behavior can be made more or less *goal-oriented* in its selection by varying the ratio of γ to ϕ (the amount of activation energy injected by the state per true proposition). For example, if $\phi = 0$, traditional backward chaining is performed (i.e., the selection is completely goal-oriented). On the other hand, the system now takes less advantage of opportunities, it is less reactive and less biased by what is currently observed and what is predicted to become true in the near future. Furthermore, it is also slowed down because the current state of the environment does not bias the action selection. Ideally we want a system that is mainly goal-oriented, but does take advantage of interesting opportunities. This can be obtained by choosing $\gamma > \phi$. The optimal ratio is of course problem dependent (more on choosing the parameter values in section 6.4).

5.2 Situation Relevance

The algorithm activates the modules that are relevant to the current situation more than the ones that are not. The processes responsible for this are the input of activation energy coming from the state of the environment and the spreading of activation energy by executable modules towards their successors (which

implements some sort of prediction of what will be true next). As already mentioned in the previous section, the advantages are that (1) the system biases its search and thereby speeds up the action selection and (2) the system is able to exploit opportunities (let its action selection be driven more by what is happening in the environment). The importance of (2) for an autonomous agent has recently been recognized by the AI community as is witnessed by the growth of interest in so-called reactive systems. The characteristic of situation-orientedness can be exploited to a higher or lesser degree by varying the parameter ϕ. Figure 4 shows the results of experiments with different ratios for the parameters γ and ϕ.

The forward spreading rules take care that a module receives activation from the state in proportion to how 'close' it is to being executable given the current state of the environment. A module is closest to being executable if it really is executable (i.e., if all its preconditions are fulfilled). For non-executable modules, 'closeness' is inversely proportional to the weighted sum of the lengths of a path from executable modules to the module itself for each of the preconditions of the module. This implies for example, that a module that has two preconditions $p1$ and $p2$ of which one, for example $p1$, cannot be made true given the current state, receives relatively less activation from the state and, therefore, has less probability of being part of a 'plan'[4].

5.3 Adaptivity

The action selection process is completely 'open'. The environment as well as the goals may change at run time. As a result, the external input/output as well as the internal activation/inhibition patterns will change reflecting the modified situation. Even more, the external influence during 'planning' or spreading activation is so important that plans are only formed as long as the influence or input/output (or 'disturbance') from the environment and goals is present.

Because of this continuous 'reevaluation', the action selection behavior adapts easily to unforeseen or changing situations. For example, if after the activation of module 'pick-up-board', the board is not in the robot's hand (e.g. because it slipped away), the same competence module becomes active once more, because it still receives a lot of activation from the competence modules that want the board to be in the robot's hand. Or if there would be a second module which can make that condition become true, than that one will be tried (because 'pick-up-board's activation level will have been reset to 0). Serendipity is another example of this ability to adapt. If a goal or subgoal would suddenly appear to be fulfilled, the modules that contributed to this goal will no longer be activated. All of these experiments have been simulated with success. Notice that such unforeseen events

[4]It may however receive a lot of activation from the goals and use that activation to urge its predecessors to make its preconditions true.

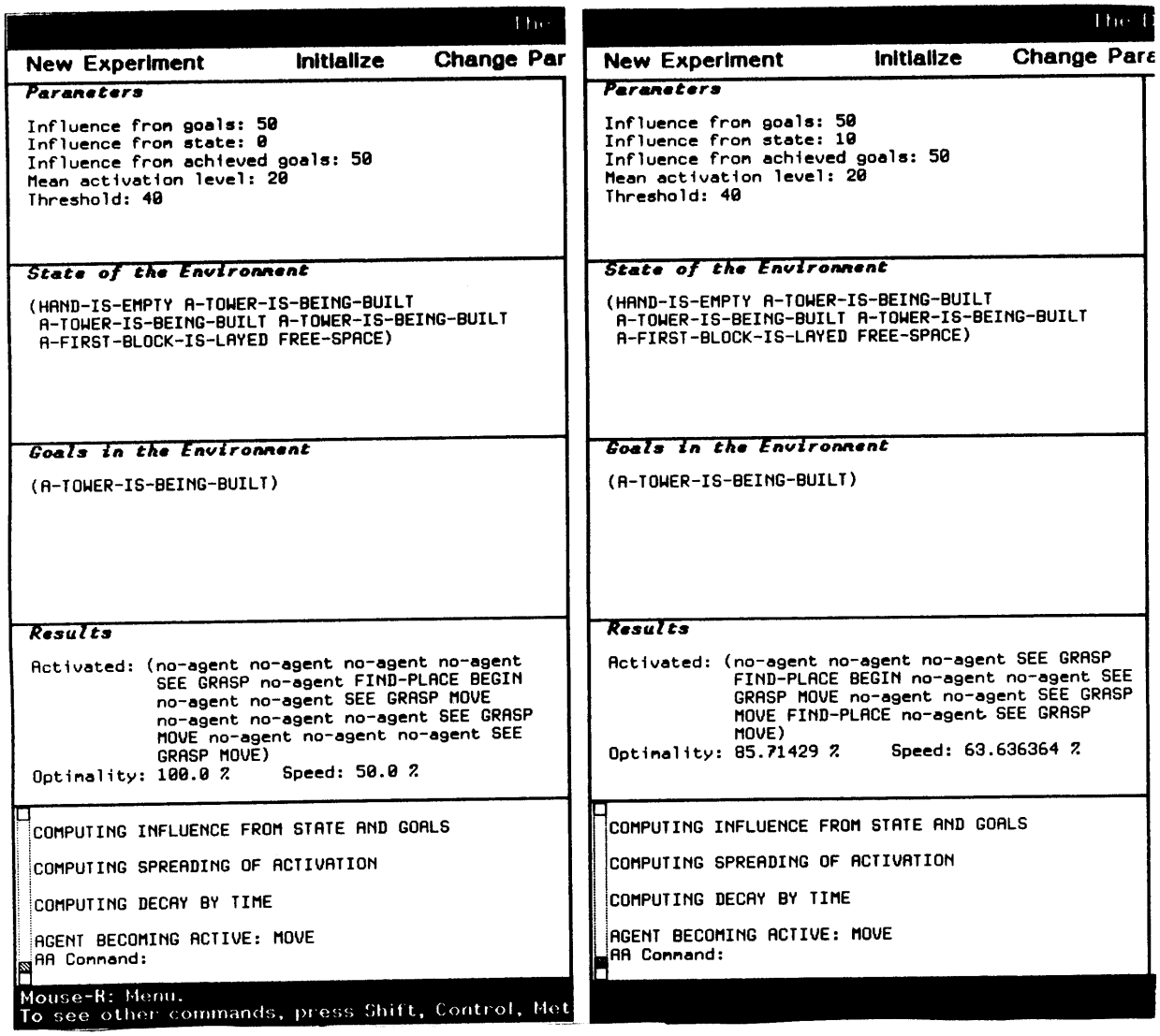

Figure 4: These results show that one can mediate between goal-orientedness of the action selection and data-orientedness by varying the ratio of γ to ϕ. In the first experiment, the network performs traditional backward chaining ($\phi = 0$). In the second experiment there is some forward spreading going on, but ϕ is still smaller than γ . The input from the state and forward spreading bias the search so that the action selection is now much faster. The resulting action selection is however less optimal (the action selection is more data-driven, which makes that actions that are not relevant to the goal may get selected, e.g. in this case, 'find-place' is activated a second time).

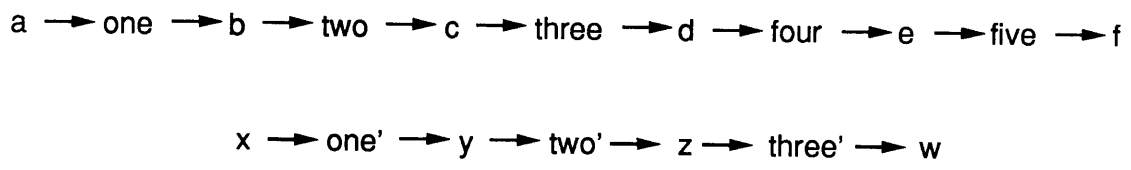

Figure 5: A toy network to test adaptivity versus bias (inertia). $y \rightarrow three$ stands for proposition y is a precondition of module *three*, while $three \rightarrow y$ stands for proposition y is in the add-list of module *three*.

do not mean that the system has to 'drop' the ongoing plan and 'build' a new one. Actually the system continuously compares the different alternatives. When some condition changes, this may have the effect that an alternative (sub-)plan becomes more attractive (more activated) than the current one.

Notice also that it is not the case that the system replans at every timestep. The 'history' of the spreading activation also plays a role in the action selection behavior since the activation levels are not reinitialized at every timestep. So just like there is a tradeoff between goal-orientedness and state-orientedness, we here have a tradeoff between adaptivity and bias towards the ongoing plan (see also next section). One can smoothly mediate among the two extremes by selecting a particular ratio of the parameters γ and ϕ versus π (the mean level of activation).

Consider as an example the modules of figure 5. The initial state is (a, x), the goal is f. After module 'one' had been active, we added w to the global goals. When γ and ϕ are relatively small in comparison with π, the internal spreading activation has more impact than the influence from the state of the environment and the global goals. The resulting action selection behavior is therefore less adaptive. Concretely in this example it means that, although for goal w the path from state to goals is shorter, the system continues working on goal f, and only after f is achieved, start working on goal w (cfr. figure 6). Again the appropriate solution lies somewhere in the middle. The parameters should be chosen such that the system does not jump between different goals all the time, but that it does exploit opportunities and adapts to changing situations.

Notice finally that the algorithm also exhibits another type of adaptivity, namely *fault tolerance*. This is a consequence of the distributed nature of the algorithm. Since no one of the modules is more important than the others, the networks are still able to perform under degraded preconditions. It is possible to delete competence modules and the network still does whatever is within its remaining capabilities. For example, when 'put-board-in-vise' is deleted or made inactive, the network comes up with a solution that does not involve this module.

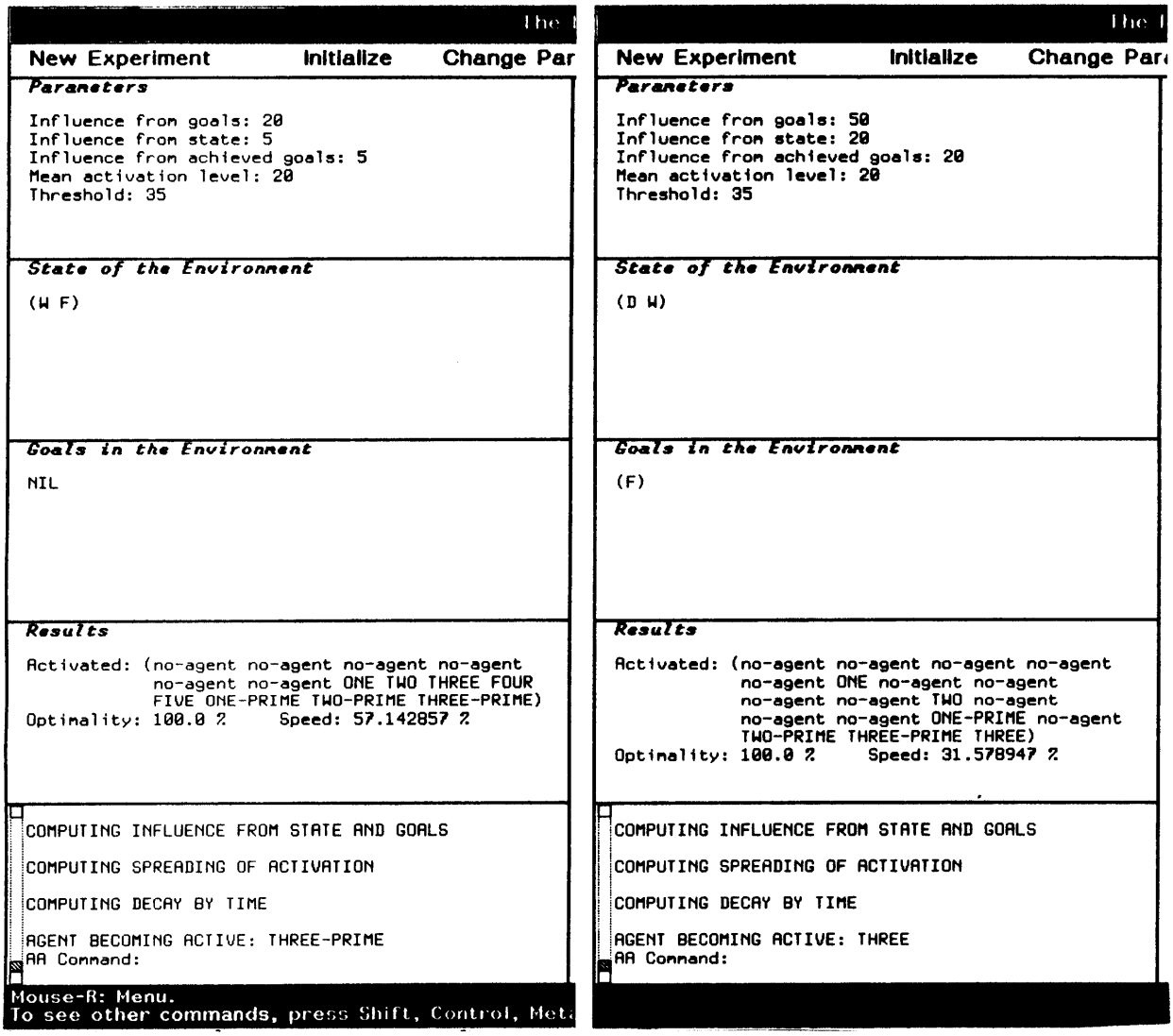

Figure 6: The action selection behavior can be made less adaptive and more biased towards ongoing plans by choosing γ and ϕ relatively small in comparison with π as in the first experiment. After module *one* had been active, we added the goal w. Although there are less modules required to achieve this goal, the system continues working on goal f. In the second experiment, the system is less biased towards ongoing goals, because γ and ϕ are relatively high in comparison with π.

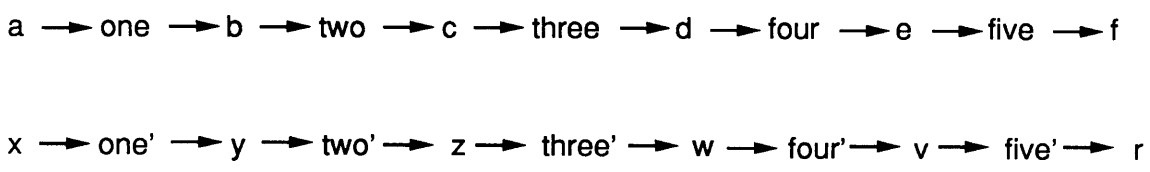

Figure 7: A toy network to test horizontal bias.

5.4 Bias to Ongoing Plans

The algorithm demonstrates an implicit bias mechanism. It favors modules that contribute to the ongoing goal and subgoals except when there is enough urge to start working on something different. The main reason bias is exhibited is that the activation levels are not reinitialized every time a module is activated. As a consequence the history of past activation spreading plays a role in the selection of action, in particular when the effect of the state and goals is relatively small in comparison with the mean activation level. But even if that is not the case, the algorithm exhibits bias towards ongoing plans. More specifically, it demonstrates two types of bias: horizontal and vertical.

1. Horizontal Bias

A first type of bias demonstrated by the action selection algorithm is the favoring of actions that contribute to the current goal (the goal on which it was working before). Given the set of modules in figure 7 and an initial state $S(0) = (a, x)$, and global goals $G(0) = (f, r)$. One to five are the competence modules necessary to achieve goal f, while one' to five' are the modules that contribute to goal r.

When simulated this network does not jump back and forth between modules that contribute to f and modules that contribute to r. Instead it starts working on one goal, completes it and then works on the other goal (cfr. figure 8). This is the case, because when either module one or one' is chosen, the distance of that path to the goals is shorter than that of the other path. Therefore, the spreading of activation backwards has a larger effect and makes sure that the started path is finished first. As the paths from state to goals grow longer, the threshold has to be increased to obtain this effect (more on the effect of the threshold in the next section).

2. Vertical Bias

A second type of bias is the favoring of actions that contribute to a 'brother' goal (a subgoal of the same overall goal). Consider the modules in figure 9. The initial state of the environment is $S(0) = (a1, c1, e1, g1, a2, c2, e2, g2)$, the goals are $G(0) = (k1, k2)$.

Again, if the threshold is high enough, this network first executes all the actions that contribute to one goal and then starts working on the other goal (cfr. figure 10). The reason is that once a predecessor of a module has been active, the node

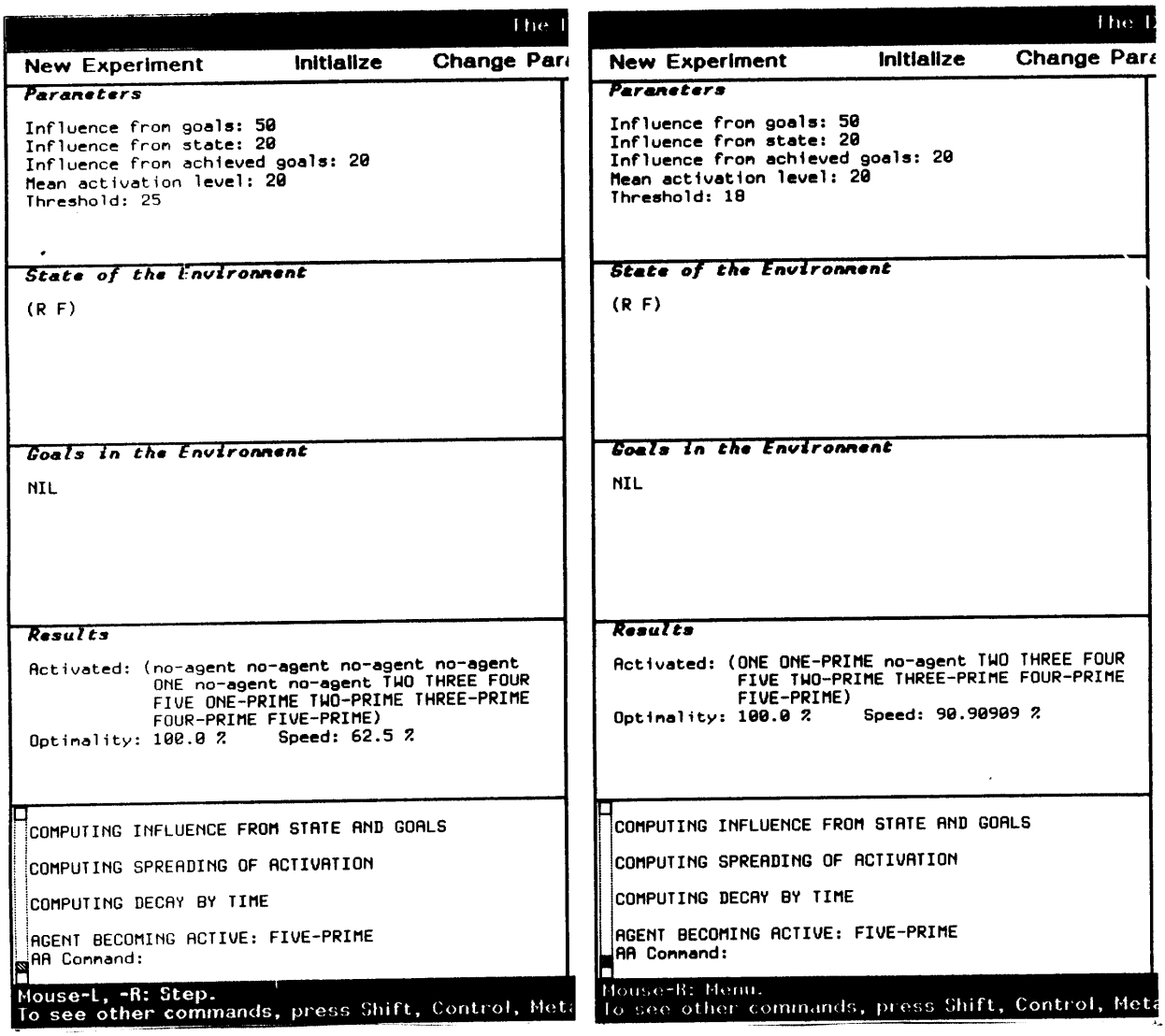

Figure 8: When the threshold is high enough, the action selection behavior exhibits a horizontal bias (left-hand experiment). When the threshold is not high enough, the system jumps between modules contributing to one goal and modules contributing to the second goal (right hand experiment).

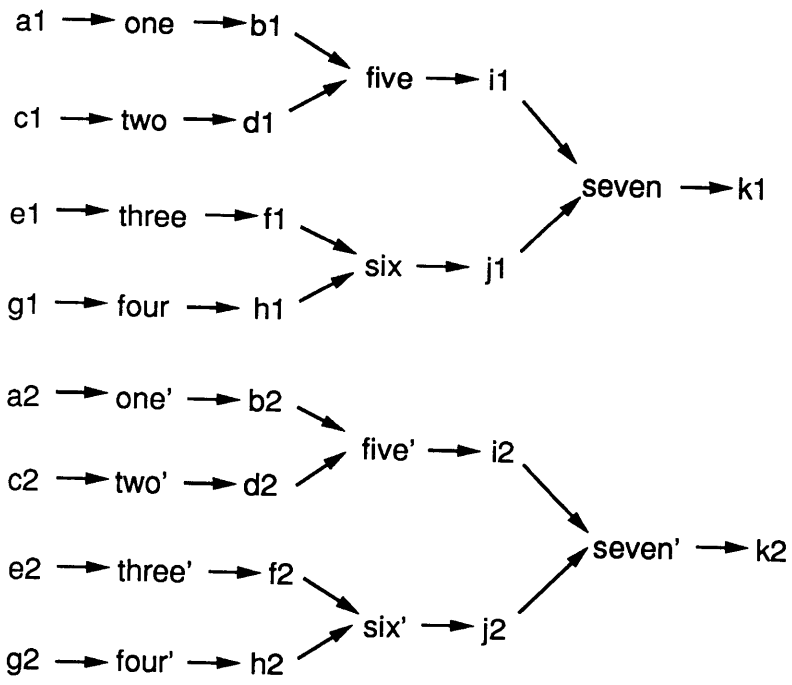

Figure 9: A toy network to test vertical bias.

itself receives more activation energy from the state of the environment. Therefore it has more activation to spread to its remaining predecessors.

As already stated in the previous section, the system can be given a higher or lesser degree of 'inertia' with respect to the changing environment and goals by selecting the ratio of the global parameters appropriately. Especially in very dynamic environments, it might be necessary to make the system adapt slower, otherwise it might never get anything done.

5.5 Avoiding Goal Conflicts

A bad ordering of actions can dramatically increase the number of actions necessary to achieve a goal, or even prevent a solution from ever being found. Any action selection algorithm should therefore to some degree be able to arbitrate among conflicting actions. Our algorithm is able to do so because of the inhibition rules. The modules in a network that undo a protected goal are weakened by a factor of δ. If δ is large enough (in particular in relation to γ and ϕ), this results in an action selection that protects global goals.

The same is true for subgoals (or preconditions of modules). Every module decreases the activation level of modules that undo its true conditions. Again this results in an action selection behavior in which 'subgoals' are protected and thereby

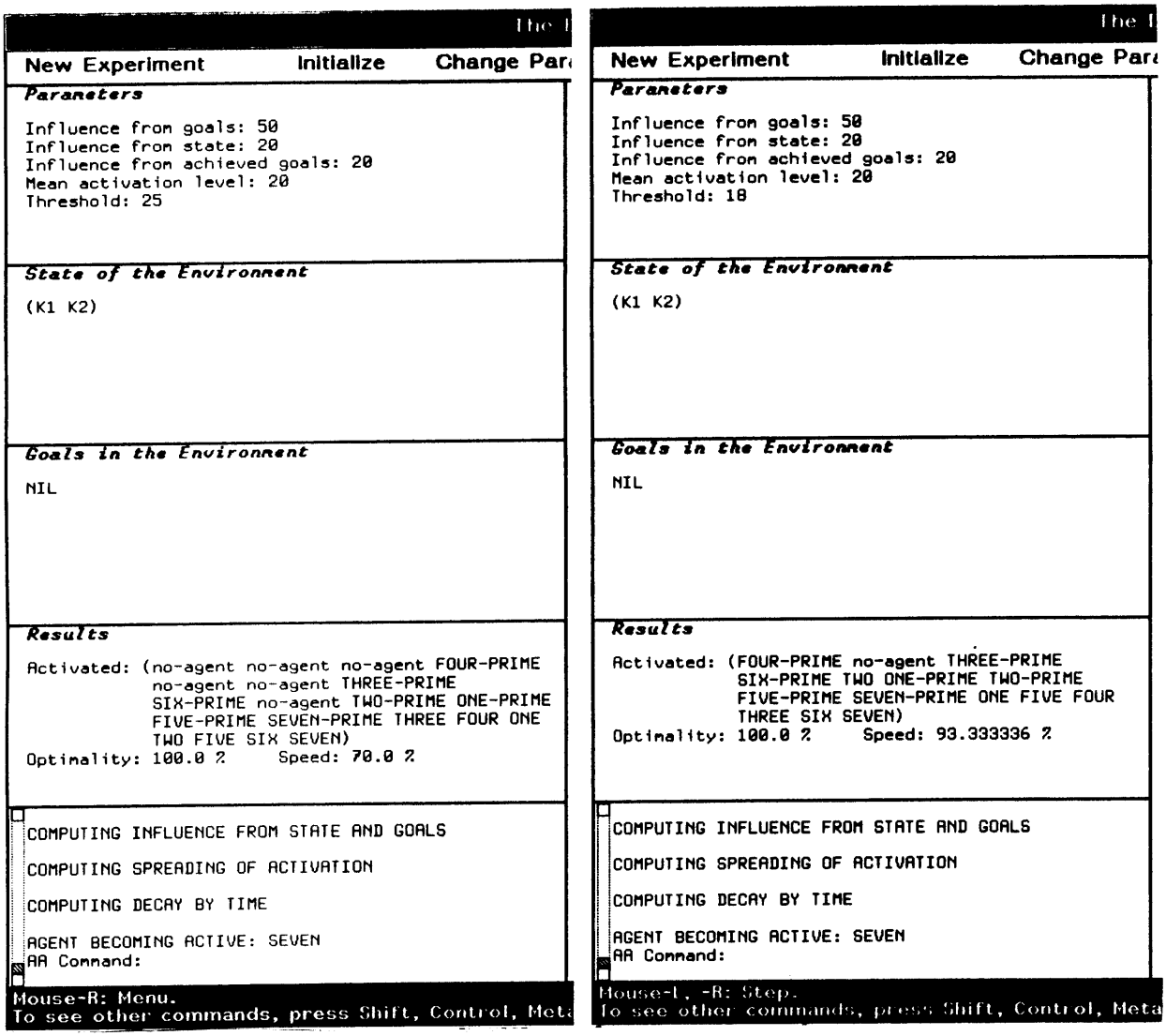

Figure 10: When the threshold is high enough, the action selection behavior exhibits vertical bias (left-hand experiment). When the threshold is not high enough, the system jumps between modules contributing to the first goal and modules contributing to the second goal (right hand experiment).

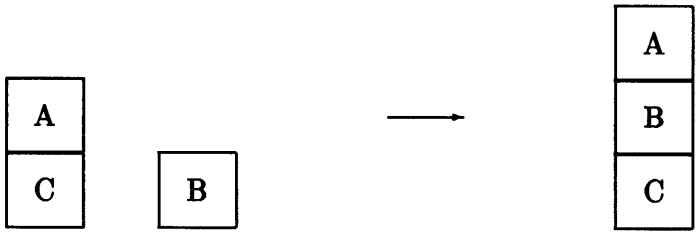

Figure 11: The classical conflicting goals example. The initial state of the world is *S(0)=(clear-a, clear-b, a-on-c)*, the goals are *G(0)=(a-on-b, b-on-c)*. The system should first achieve the goal b-on-c and then the goal a-on-b. It is tempted however to immediately stack a onto b, which may bring it in a deadlock situation (not wanting to undo the already achieved goal).

```
(defmodule stack-a-on-b
  :condition-list '(clear-a clear-b)
  :add-list '(a-on-b clear-c)
  :delete-list '(clear-b a-on-c))
(defmodule stack-b-on-c
  :condition-list '(clear-c clear-b)
  :add-list '(b-on-c clear-a)
  :delete-list '(clear-c b-on-a))
(defmodule take-a-from-c
  :condition-list '(clear-a a-on-c)
  :add-list '(clear-c)
  :delete-list '(a-on-c))
```

Figure 12: Some of the modules involved in the blocks world domain.

goal conflicts are avoided. To illustrate how this happens, we reimplemented the classical anomalous situation example from the blocks world (Sussman, 1975). Figure 11 illustrates the problem. Figure 12 shows some of the competence modules involved in this example.

Figure 13 and 14 show the results obtained. In the first experiment δ has the same value as γ which is far greater than ϕ. The result is that the inhibition of 'stack-a-on-b' by 'stack-b-on-c' for condition 'clear-b' is far more important than its activation by the state. Because of this, the module 'take-a-from-b' dominates over 'stack-a-on-b', despite the fact that the latter one achieves a goal. If δ is not high enough (as in the second experiment), the urge to fulfill the goal 'a-on-b' dominates over the urge to avoid 'clear-b', so that the system does start by stacking a on b. It is however still able to restore the situation and obtain the two goals, since the influence from the protected goals is not high enough to keep the system from undoing the achieved goal 'a-on-b'. Again, a balance has to be found

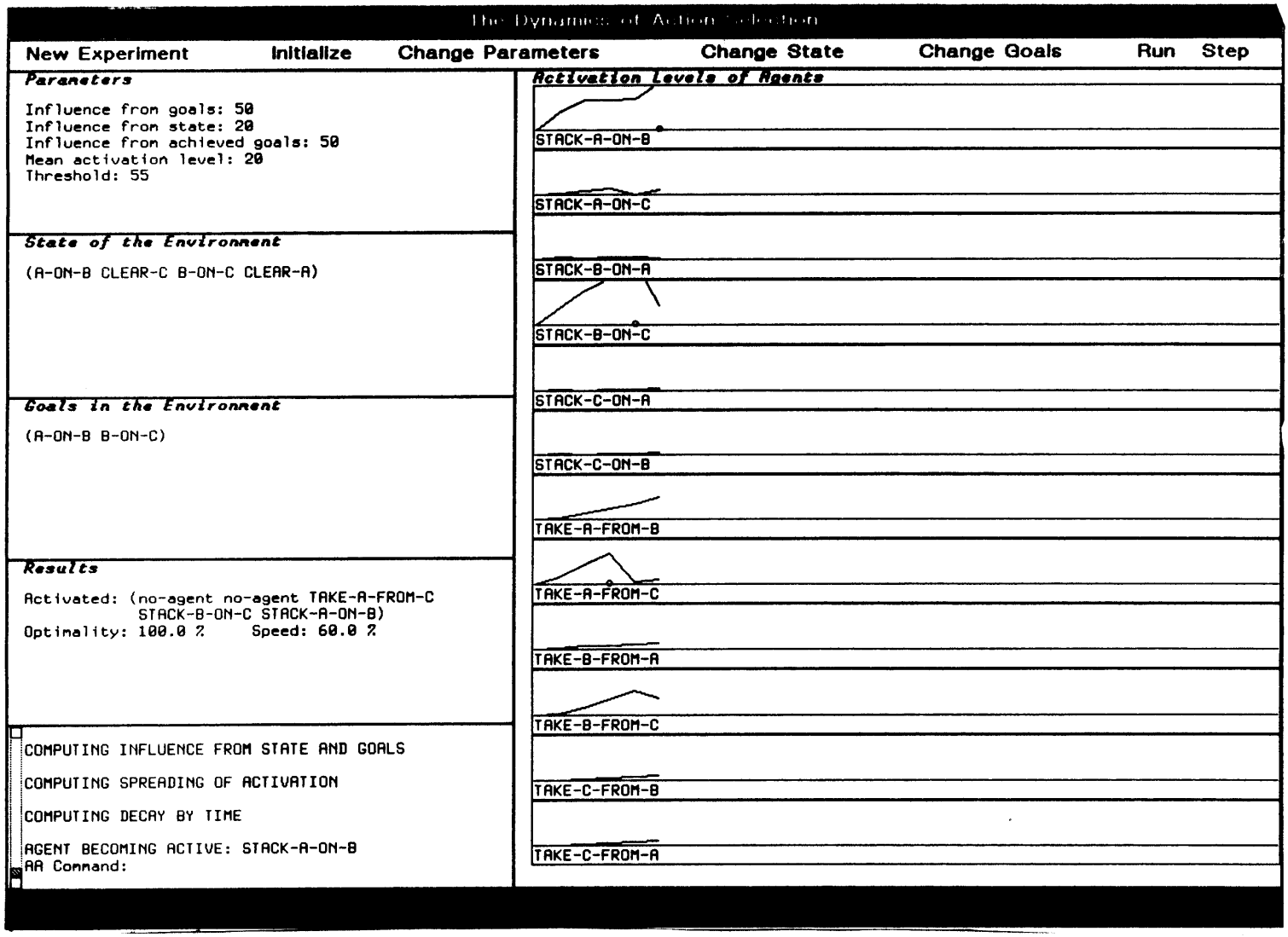

Figure 13: When the influence from protected goals and the threshold are high enough, the system is able to avoid problems with conflicting goals.

between not caring about goal conflicts at all and being so rigid as to never undo an achieved (sub-) goal, thereby risking deadlocks.

5.6 Thoughtfulness

A network only looks ahead in a local neighborhood (in time) which is determined by the threshold θ. The behavior can be made more or less *thoughtful* by increasing the threshold θ. This makes the spreading activation process go on for a longer time before a specific action is selected. As such, it allows the network to look ahead further, thereby avoiding local maxima (in time) of activation levels. For example, in the blocks-world example above, the module 'stack-a-on-b' initially has the highest activation level (since it receives direct input from both the state and the goals). The threshold has to be put high enough to avoid that this module is chosen right away, so that the network can go on taking into account the conflicts

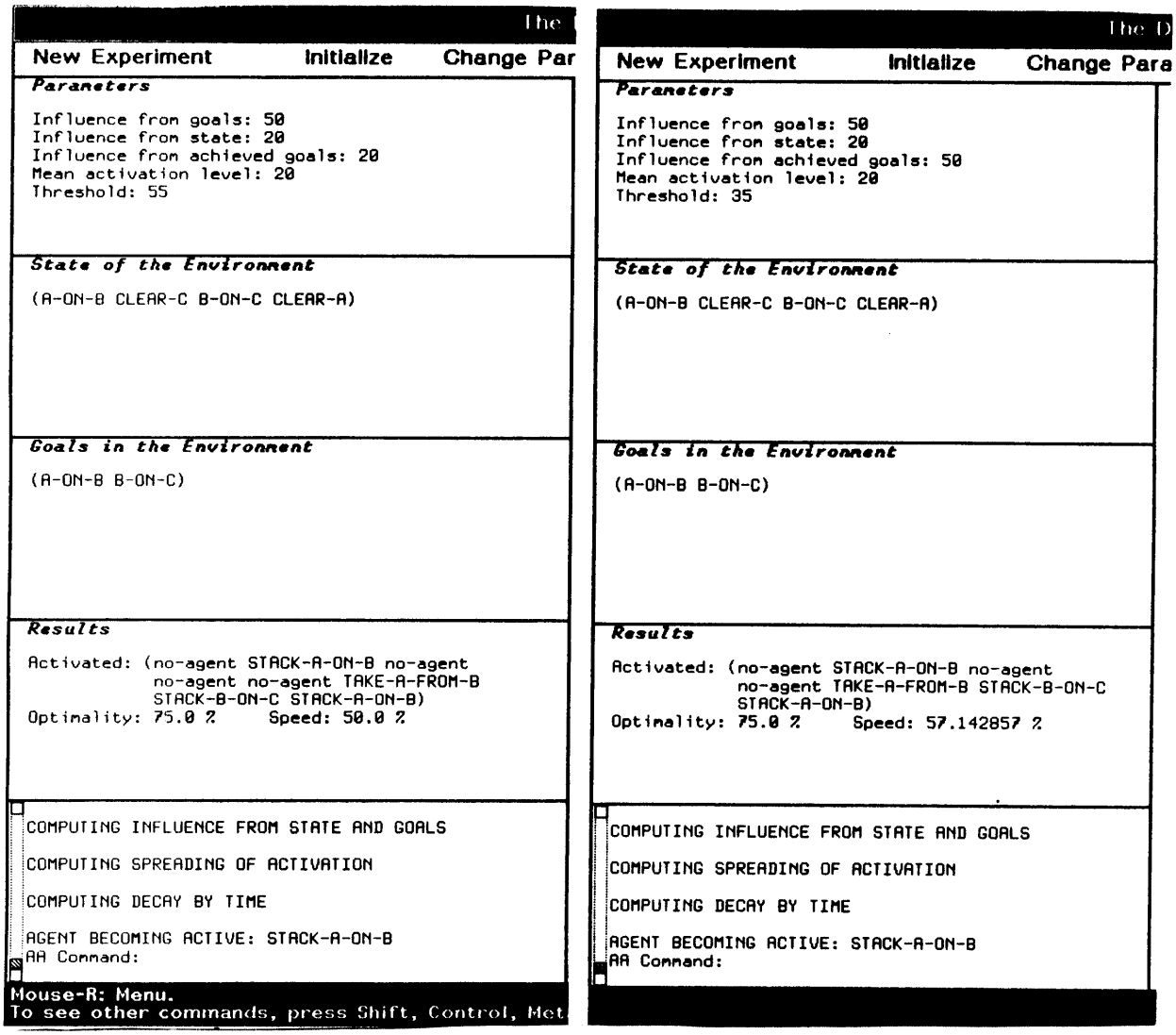

Figure 14: In both these experiments the system reacts opportunistically, not taking into account conflicting goals. In the first experiment, the parameter γ is low, so that the system is not very sensitive to goal-conflicts. In the second experiment, the threshold is not high enough, so that the system chooses a local maximum.

among modules.

Ideally, we would like to set the threshold to a very high value (for example equal to the total activation of the whole network). This would guarantee that the spreading activation process goes on long enough so that the 'optimal' action can be selected. The problems with putting the threshold high are first, that the action selection process would require too much time (especially for an agent operating in a rapidly changing environment) and second, that the result would be that the agent would get bogged down trying to take into account the effects of actions it might take in the far future. This is most probably a wasted effort in an unpredictable environment. Therefore we do want the agent only to look ahead to the near future. The desired amount of looking ahead for a particular application can be obtained by choosing a proper value for the threshold.

5.7 Speed

The counterpart of thoughtfulness is speed. The action selection behavior can be made *faster* by varying the threshold θ as explained above. The resulting action selection is however less 'thoughtful', which means that it is less goal-oriented, less situation oriented, that it takes conflicting goals less into account and that it is less biased towards ongoing plans. Nevertheless. it may sometimes be important to react fast or it may be a wasted effort to be very thoughtful (i.e., make a lot of plans and predictions).

Fortunately, the algorithm is not complex, so that it allows speed to be obtained without sacrificing too much thoughtfulness. The algorithm does however perform some sort of 'search' through a network from goal modules to executable modules, so one could argue that the algorithm suffers from the same problems as traditional AI search. More specifically, that the efficiency necessarily goes down as the number of modules involved in a plan grows (the so-called 'combinatorial explosion' problem). Nevertheless, it is important to take the following counterarguments into consideration:

- The search that is going on here is of a very different nature. Actually, it resembles marker passing algorithms more than the AI notion of search. The system does not construct a search tree, nor does it maintain a current hypothetical state and partial plan. In addition, it evaluates different paths in parallel, so that it does not have to start from scratch when one path does not produce a solution, but smoothly moves from one plan to another. As a result, the computation the algorithm performs is much less costly.

- The system does not 'replan' completely at every timestep. The algorithm does not reinitialize the activation-levels to zero whenever an action has been taken. This implies that it may take some time to select the first action to

execute, but from then on, the network is biased towards that particular situation and set of goals. This means that it will take much less time for the following actions to be selected, in particular when little has changed in the meantime with respect to the goals or current situation.

- We believe that for real autonomous agents (e.g., mobile robots) the networks will grow 'larger' instead of 'longer', because typically, the agent will have more tasks/goals instead of having tasks/goals that require more actions to be taken (and therefore more 'planning'). Also, large subparts may exist in the network that appear to be unconnected. As a result, the efficiency of the system will not be affected so much. Even if some paths from state matchers to goal achievers would be very long, the system would still come up with an action because it does not await a convergence in the activation levels and decreases the threshold with time. The selected action might however be non optimal.

- The same simple spreading activation rules are applied to each of the modules. In addition, there are only local, fixed links among modules. This opens interesting opportunities for a parallel implementation, which would imply a considerable speed up.

6 Discussion

There are a number of limits to the algorithm as it is now. The main ones are listed below.

- The language provided to describe the input-output relationship of a competence module is oversimplified. There is no way to work with abstractions, neither can variables be used.

- A network does not maintain a record of its past 'search'. As such the same planning mistake can be made over and over again in the same plan, making the system loop.

- It is not yet clear how, given a specific application, one can select values for the global parameters that produce the desired action selection behavior.

In the remainder of this section we discuss the importance of these limits and sketch solutions to those that represent important limitations. The proposed solutions resonate with the current philosophy and the merits it has. The implementation of these solutions will be the main concern of our future research.

6.1 Variables

The algorithm does not incorporate classical variables and variable-passing. As a matter of fact, a lot of its advantages would disappear if they would be introduced. For example, one reason a lot of search is eliminated is exactly because there are no variables in the algorithm. A first implication of the absence of variables is that one cannot specify goals using variables (e.g. goto-location(x,y)). A second implication is that all modules/operators of the domain have to be instantiated beforehand, which means that a node has to be created for every parameter.

We try to avoid the need for variables altogether by using only so-called *indexical-functional aspects* to describe relevant properties of the immediate environment (Agre & Chapman, 1987). The main idea here is that internal representations of objects in the environment are in terms of the purposes and circumstances of the agent. The module 'spray-paint-self' for example only has to be instantiated with one parameter, namely 'the-sprayer-I-am-holding-in-my-hand'. Because of this, it is not necessary to create new operators/modules for every new object that is introduced in the world. There is no exhaustive combination of operators and objects.

The idea of indexical-functional aspects is particularly interesting for autonomous agents because it does not make unrealistic assumptions about what perception can deliver. In particular, it does not demand that perception can produce the identity and exact location of objects. The absence of variables does constrain the language one can use to communicate with the system, but not in a too strong way. All it requires is a new way of thinking about how to tell an agent what to do. More specifically, one does not use unique names of objects when specifying goals. Instead goals are specified in terms of indexical or functional constraints on the objects involved. For example, one would not tell the agent to go to location (x,y), but one would tell the agent that the goal is to be in a location that is a doorway (a small area where it is able to 'go through' a wall).

6.2 Handling Loops

A problem with the current algorithm is that loops in the action selection may emerge. They only occur very rarely and spring from the fact that the system does not maintain a history of what it did before. It is questionable whether a solution to such impasses should be built in. The hypothesis could be adopted that in a real environment the state and goals will change anyway after some time $\triangle t$ that is very small. This changes the spreading activation patterns and therefore gets the network out of its impasse. If we insist on avoiding (even temporal) impasses, this cannot be guaranteed by a careful selection of the parameters. One very simple solution however could be to introduce some randomness in the system. Another solution might be to use a second network to monitor possible loops in the

first network and take actions whenever this happens. Finally, we could implement some *habituation* mechanism for some or all of the modules. This mechanism would take care that every time a module is activated, it is less likely to become active in the future (i.e., have local thresholds that vary over time).

6.3 Selecting the Parameters

The global parameters to a large degree determine the effectiveness and characteristics of the action selection behavior. It is still an open question how the values for these parameters should be selected. They are very problem dependent, not only because every problem area requires different degrees of goal-orientedness, situation-orientedness, speed, adaptivity, etc. But also because the size and structure of the network also determines these characteristics. For example, in an application with a very big network, the threshold has to be put higher to obtain the same results. At the moment we tune the parameters by hand during a series of experiments. We plan to build a second network of competence modules that would look at the results of the first one and tune its parameters so as to obtain the action selection characteristics specified by the user.

7 Related Work

The introductory section already discussed how this work relates to connectionism and traditional AI. The main difference with the former being that more structure and competence is built in, the difference with the latter being that classical search is avoided. The remainder of this section compares this work to so-called 'reactive systems', to distributed AI and to other hybrid systems.

7.1 Reactive Systems

The approach is related to the so-called 'reactive systems' (Georgeff & Lansky, 1987) (Firby, 1987) (Kaelbling, 1987) (Rosenschein & Kaelbling, 1987) (Schoppers, 1987) (Agre & Chapman, 1987) (Sanborn & Hendler, 1987). The emphasis in these architectures is on a more direct coupling of perception to action, distributedness and decentralization, dynamic interaction with the environment and inherent mechanisms to cope with resource limitations and incomplete knowledge. They deemphasize deliberation (or 'thinking' in general) and internal models. The main difference between our algorithm and these systems is that we neither 'prewire' nor 'precompile' the control flow. The arbitration among modules is a run-time process which differs according to the goals that are given to the system and the situation the system finds itself in. It therefore constitutes a simpler, more distributed and more general solution to the problem of action selection.

7.2 Distributed AI

The difference between this work and the bulk of work in distributed planning (Bond & Gasser, 1988) (Huhns, 1987) as well as with the work on black-board systems (Hayes-Roth, 1979), is that in the latter planning modules communicate among themselves on a much higher level. They communicate using a language, sometimes debate and negotiate among one another or even reason about each other. The problem-specific needs for a communication language therefore constitutes the major barrier for the widespread applicability of these techniques. The algorithm presented in this paper makes integration of different modules in one system easier because the communication among modules is reduced to a minimum and happens on an information-scarce level (only numbers are being communicated). Furthermore, modules do not have to share a global internal model or global blackboard. They are said to communicate 'through the world' (Brooks, 1986).

7.3 Hybrid Systems

Finally, this algorithm is related to some of the hybrid systems that have been built for planning and decision making. Hendler (1988) describes a hybrid system in which a massive parallel component is used to provide heuristic information to a classical AI planner. A marker propagating network guides the classical planner towards more relevant plans. (Lehnert, 1987) describes a hybrid system that uses a stack and copy mechanism for control and numerical relaxation over a structured network for smooth decision making. The difference with the algorithm presented here is that in both these systems the control is still hierarchical and centralized, and might therefore turn out to be too inflexible for use in autonomous agents operating in a dynamic environment.

8 Conclusions

The results reported upon in the paper demonstrate the feasibility of using an activation/inhibition dynamics among competence modules to solve the problem of action selection for an autonomous agent operating in a dynamic world. Such a scheme has particular advantages over traditional, deliberative hierarchical methods. The price to pay is that the actions selected might be less rational. However, the algorithm provides global controls which one can use to tune the action selection behavior along several criteria, such as thoughtfulness/rationality versus speed, goal-orientedness versus data-orientedness, and adaptivity versus bias to ongoing goals.

9 Acknowledgements

Nils Nilsson, Marcel Schoppers, Dan Shapiro and Michael Braverman provided very valuable criticisms and suggestions. Marvin Minsky, Rodney Brooks, Luc Steels, J.R. Anderson and Jerome Feldman provided inspiration. Jan Torreele, Piet Spiessens and Tony Bell proofread the paper. Supported by Siemens with additional support from the University Research Initiative under Office of Naval Research contract N00014–86–K–0685, and the Defense Advanced Research Projects Agency under Office of Naval Research contract N00014–85–k–0124. The author is a research associate of the Belgian National Science Foundation. She currently holds a position as visiting professor at the M.I.T. Artificial Intelligence Laboratory.

10 References

Agre, P. & Chapman, D. (1987) Pengi: An Implementation of a Theory of Activity. Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87. Morgan Kaufmann, Los Altos, California.

Bond, A. & Gasser, L. (1988) Readings in Distributed Artificial Intelligence. Morgan Kaufmann, San Mateo, California.

Brooks, R. (1986) A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation. Volume RA-2, Number 1.

Chandrasekaran, B., Goel, A. & Allemang, D. (1988) Connectionism and Information Processing Abstractions, AI-magazine, Vol. 9, No. 4.

Charniak, E. & Mc Dermott, D. (1985) Introduction to Artificial Intelligence. Addison-Wesley.

Firby, R. (1987) An Investigation into Reactive Planning in Complex Domains. Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87. Morgan Kaufmann, Los Altos, California.

Georgeff, M. & Lansky, A. (1987) Reactive Reasoning and Planning. Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87. Morgan Kaufmann, Los Altos, California.

Hayes-Roth, B. et. al. (1979) Modeling Planning as an Incremental Opportunistic Process. Proceedings of IJCAI-79, Tokyo, Japan.

Hendler, J. (1988) Integrating Marker-Passing and Problem Solving, A Spreading Activation Approach to Improved Choice in Planning. Lawrence Erlbaum Ass.,

Hillsdale, New Jersey.

Huberman, B. & Hogg, T. (1987) Phase Transitions in Artificial Intelligence Systems. AI-Journal, Volume 23, Number 2.

Huhns, M. (1987) Distributed Artificial Intelligence. Pitman, London.

Kaelbling, L. (1987) An Architecture for Intelligent Reactive Systems. Reasoning about Actions and Plans: Proceedings of the 1986 Workshop. Morgan Kaufmann, Los Altos, California.

Laird, J., Rosenbloom, P. & Newell, A. (1986) Chunking in SOAR: The Anatomy of a General Learning Mechanism. Machine Learning. Volume 1, Number 1. Kluwer Academic Publishers.

Lehnert, W. (1987) Case-Based Problem Solving with a Large Knowledge-Base of Learned Cases. Proceedings of the AAAI-87 Conference, Seattle, Washington.

Maes, P. (1989) The Dynamics of Action Selection. Proceedings of the IJCAI-89 conference, Detroit.

Minsky, M. (1986) The Society of the Mind. Simon and Schuster, New York, New York.

Rosenschein, S. & Kaelbling, L. (1987) The Synthesis of Digital Machines with Provable Epistemic Properties. In J.F. Halpern (editor), Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge. Morgan-Kaufmann, Los Altos, California.

Sanborn, J. and Hendler, J. (1988) A Model of Reaction for Planning in Dynamic Environments. International Journal of AI in Engineering, 1988. Special Issue on Planning.

Schoppers, M. (1987) Universal Plans for Reactive Robots in Unpredictable Environments. Proceedings of IJCAI-87, Milan, Italy.

Simon, H. (1955) A Behavioral Model of Rational Choice. Quarterly Journal of Economics, 69: 99-118.

Steels, L. (1989) Connectionist Problem Solving, an AI Perspective. Connectionism in Perspective, Eds: Pfeiffer R., Schreter Z., Fogelman F. and Steels L., Elsevier, Amsterdam.

Sussman, G. (1975) A Computer Model of Skill Acquisition. Elsevier Publishers. New York.