Adaptive Action Selection

Pattie Maes
MIT Media-lab (E15-495)
20 Ames Street
Cambridge, MA 02139
pattie@media.mit.edu

Abstract

In earlier papers we presented a distributed model of action selection in an autonomous intelligent agent (Maes, 1989a, 1989b, 1991a, 1991b). An interesting feature of this algorithm is that it provides a handful of parameters that can be used to tune the action selection behavior of the algorithm. They make it possible, for example, to trade off goal-orientedness for data-orientedness, speed for quality, bias (inertia) for adaptivity, and so on. In this paper we report on an experiment we did in automating the tuning and run-time adaptation of these parameters. The same action selection model is used on a meta-level to select actions that alter the values of the parameters, so as to achieve the action selection behavior that is appropriate for the environment and task at hand.

Introduction

An important problem to be addressed when modeling an intelligent agent is the problem of *action selection*: what mechanism can an agent use to determine what to do next, given that it has a repertoire of actions it can engage in, wants to achieve certain goals (or has certain motivations) and is facing a particular situation. Computational models of action selection are important both for producing actions in an artificial agent (e.g. an autonomous mobile robot) as well as for understanding the action selection behavior of biological agents (human and animal). In previous papers we argued that none of the models of action selection that have been presented in the literature on AI present a satisfactory solution to the problem (Maes, 1991a, 1991b). Two classes of approaches can be identified: traditional *planners* and *reactive systems*. The former are not satisfactory because they do not produce fast (re)action, are completely goal-driven and not adaptive. The problem with reactive systems is that they are completely sensor-data driven, do not have a notion of goals, cannot anticipate or predict the outcome of actions and are hard to design.

In (Maes, 1989a, 1989b, 1991a, 1991b) we presented an action selection algorithm which tries to combine the best of previous models. This algorithm produces fast action in a tight connection with the environment, while providing a notion of goals and run-time arbitration. Just like traditional planners, our algorithm uses an explicit description of the conditions and expected effects of actions (cf. STRIPS operators). But instead of implementing action selection as a deliberative, sequential reasoning process, we use this information to construct a network, in which action selection is an emergent property of an activation/inhibition dynamics among the nodes (operators/actions). The algorithm provides a distributed solution to the problem of action selection through the use of a connectionist computational model on a symbolic representation. The results of experiments prove that our algorithm does arbitrate among actions, and implements a form of prediction, planning and anticipation.

One of the interesting properties of the algorithm is that its behavior can be tuned by hand according to the requirements of a particular application. The algorithm has five global, numerical parameters which make it possible to make the action selection more goal-driven versus more data-driven, faster versus more cautious, biased towards the recent history versus adaptive, and more or less sensitive to goal and action conflicts. This is useful because different tasks and applications require different action selection behavior. For example, if an agent has very unreliable sensors, we want it to bias its action selection towards the past history of action selection. Or if the environment is very dynamic, we want the action selection behavior to be very "reactive" and data-driven.

Until recently, we set the values of these parameters by hand by means of a generate and test iterative process. This solution is far from ideal, first of all, because this process turned out to be very difficult in certain cases, and second, because the parameters did not adapt to changes in the task or environment at run time (unless the programmer would go through the same process again). In this paper we describe an extension of the algorithm which allows the system to autonomously adapt the parameters to the characteristics of the task and environment. This functionality is

achieved by using the same action selection algorithm on a "meta-level". A meta-network was constructed which consists of actions that observe the behavior of the object-level network and control its parameters so as to achieve some preset goals (in terms of performance).

The Basic Algorithm

This section summarizes the basic action selection model and algorithm, without the meta-level control. More elaborate descriptions of the algorithm and its results can be found in (Maes, 1989b, 1991a, 1991c). An intelligent agent is viewed as a collection of *competence modules*. Competence modules are like the (mindless) agents populating the Society of Mind (Minsky, 1986): they are autonomous modules that are expert in achieving a particular competence, such as grasping a cup. Every competence has an associated *activation level*, which is a real number. A competence module further also has a set of *conditions* which have to be observed in order for the competence module to be *executable*. And finally, a competence module has an *add-list* and *delete-list* which describe the expected effects on the state of the world that the module has. An executable competence module whose activation level surpasses a preset threshold can be selected (become active). This means that a set of processes start running which try to "realize" the competence (certain real world actions are performed at that moment). Consider as an example the pick-up-board competence module described below.

```
(defmodule PICK-UP-BOARD
  :condition-list '(board-within-reach hand-empty)
  :add-list '(board-in-hand)
  :delete-list '(board-within-reach hand-empty)
  :processes <a set of processes that implement
              the action of picking up the board>)
```

An agent also has a set of *goals* (or motivations), which have an associated number representing the strength of that goal at a particular moment in time. For example, the agent can have the goal of sanding the board with associated strength 40 and the goal of spray painting itself with strength 90. An agent also has a set of *protected goals*, which are goals that are achieved and that should remain achieved. E.g. once the board has been sanded, this becomes a protected goal. The set of goals and their associated strengths vary over time as the agent takes actions. Finally an agent has a set of *perceptual conditions* (or sensor data) that it observes at a particular moment in its environment

The different competence modules of an agent are linked in a network through "predecessor", "successor" and "conflicter" links. In this paper we assume that these links are known. The idea is that these links among competence modules are either innate (i.e. programmed) or learned through experience (Maes & Brooks, 1990). There is a predecessor link from competence module A to competence module B (competence module A has competence module B as predecessor) if competence module B (through the actions it takes), makes certain conditions of competence module A come true (the intersection of B's add-list with A's condition-list is non empty). For example, 'pick-up-board' may make the condition of 'sand-board' that the board has to be in the hand of the agent, become true. Second, there is a matching successor link in the opposite direction for every predecessor link. Finally there is a conflicter link from competence module A to competence module B (competence module B is said to conflict with competence module A) if competence module B makes a certain condition of competence module A undone (the intersection of B's delete-list with A's condition-list is non empty). For example, 'put-down-board' may make the condition of 'sand-board', that board has to be in the hand, untrue.

The intuitive idea behind the action selection mechanism is that competence modules use the links of the network to activate and inhibit each other (respectively increase and decrease each other's activation level), so that after some time the activation energy accumulates in the competence module that represents the "best" choice, given the current situation and motivational state of the agent. Once the activation level of a competence module reaches a certain threshold, it may be selected, and its processes start operating. The pattern of spreading activation among competence modules, as well as the input of new activation energy into the network is determined by the current situation and the motivational state of the agent. The input of new activation energy into the network of competence modules is defined as follows:

- *Activation by the Current Situation.* The currently observed perceptual conditions (sensor data) increase the activation level of the competence modules that have those conditions. For example, if the agent observes a board within its reach, then the activation level of the 'pick up board' competence module is increased.
- *Activation by the Goals.* The goals of the agent increase the activation level of the competence modules that might achieve them (have the goal in their add-list) with an amount which is relative to the strength of the goal. For example, the activation level of 'sand-board' is increased with some amount proportional to the strength of the goal 'board-sanded'.
- *Inhibition by the Protected Goals.* Finally the protected goals of the agent decrease the activation level of the competence modules that might undo them (have the protected goal in their delete-list) with an amount that is relative to the strength of the goal.

These processes are continuous: there is a continual flow of activation energy towards the competence

modules whose condition list partially matches the current situation and towards the modules whose add-list matches the goals. The modules that would undo protected goals continuously weakened. If the situation or the motivational state changes (e.g. the perceived conditions change or the strengths of the goals change), the input of activation energy automatically flows to other competence modules. Besides the impact on activation levels from the current situation and goals, competence modules also activate and inhibit each other. Competence modules spread activation along their links as follows:

- *Activation of Successors.* An executable competence module spreads activation forward. It increases (by a fraction of its own activation level) the activation level of its successors. Intuitively, we want these successor competence modules to become more activated because they are "almost executable", since more of their conditions will be fulfilled after the competence module has become active. For example, if the 'pick up board' competence module is executable (i.e. a board is withing reach of the agent and it's hand is empty), this will increase the activation level of its successor, the 'sand-board' competence module.
- *Activation of Predecessors.* A competence module that is not executable spreads activation backward. It increases (by a fraction of its own activation level) the activation level of its predecessors. Intuitively, a non-executable competence module spreads to the competence modules that can fulfill its conditions that are not yet true, so that the competence module itself may become executable afterwards. For example, if 'sand-board' is not executable because the agent is not holding the board, it will spread activation energy to 'pick up board' to encourage this competence module to become active (so that afterwards 'sand-board' can become active).
- *Inhibition of Conflicters.* Every competence module (executable or not) decreases (by a fraction of its own activation level) the activation level of its conflictors. Intuitively, every competence module will try to prevent a competence module that undoes one of its true conditions from becoming active. For example, the 'sand-board' competence module will decrease the activation energy of its conflicter 'put-board-down', because the latter would undo the condition that the board has to be in the hand.

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 current situation and goals on the activation level of a competence module is computed.
2. The way the competence module activates and inhibits related competence modules through its successor links, predecessor links and conflicter links is computed.
3. A decay function ensures that the overall activation level remains within some boundaries.
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 the 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. Once a competence module has been activated (it actually performs its actions at that moment), its activation level is reset to 0 (but it may quickly increase again, since the spreading activation process goes on continuously).

We have evaluated the algorithm empirically by performing a wide series of experiments using several example applications. The networks cannot be said to show a 'jump-first think-never' behavior. They do exhibit planning or "reasoning" capabilities. The effects of a sequence of actions are considered before actually embarking on its execution. If a sequence of 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 modules is obtained through the input from the goals and the backward spreading of activation. Situation relevance and opportunistic behavior are achieved through the input of the sensor data and the forward spreading of activation. 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 are favored by the remains of the past spreading activation patterns.

The Need for Adaptive Action Selection

In the recent past, researchers have been arguing about different action selection models, trying to convince their peers that one model is more appropriate than another (AI-magazine, 1990). For example, advocates of the traditional planning model argue that reactive systems are not very "thoughtful" because they do not model and reason about the consequences of actions. They argue that such systems would fail in "critical" environments such as a nuclear power plant. On the

other hand, advocates of the reactive systems model claim that traditional planners are slow, brittle and not flexible and will fail in environments that are very dynamic. We believe that the desired characteristics for an action selection model differ from application to application. The degree to which an environment or task is "critical' (the cost of non-optimal actions) determines how much prediction is desired. The degree to which an environment is "dynamic" (changes rapidly) determines how reactive or fast the action selection has to be. The degree to which an environment is predictable, determines the usefulness of internal models, and so on.

Therefore any action selection model that aims to be "generally useful" should be tunable to the characteristics of an environment and task. This is possible for the action selection algorithm that we described above. Its behavior can be tuned to the characteristics of a particular application by a handful of global parameters (Maes, 1989b, 1991a):

- *Threshold* θ. This parameter determines how much activation energy has to be accumulated by a module before it can be selected. By varying θ one can tradeoff *speed* for *thoughtfulness*. The higher θ, the longer the activation spreading goes on before a module has accumulated enough activation energy. As such, it allows the network to look ahead further, thereby avoiding local maxima (in time) of activation levels.
- *Goal-orientedness* γ. This parameter determines the strength of the backward spreading of activation energy. It determines (i) how much activation energy is put into the network by the goals (as opposed to the sensor data) and (ii) what the (global) weight is of the predecessor links. By varying γ the action selection behavior can be made more or less goal-driven.
- *Data-orientedness* ϕ. This parameter determines the strength of the forward spreading of activation energy. It determines (i) how much activation energy is put into the network by the sensor data (as opposed to the goals) and (ii) what the (global) weight is of the successor links. By varying ϕ the action selection behavior can be made more or less data-driven.
- *Goal-conflict sensitivity* δ. This parameter determines the strength of the inhibitory spreading of activation. It determines (i) how much activation energy is taken away from the network by a protected goal (as opposed inserted by a goals) and (ii) what the (global) weight is of the conflictor links. By varying δ the action selection behavior can be made more or less sensitive to goal conflicts. For example, if δ is larger than γ, the system cares more about avoiding goal conflicts than about achieving goals.
- *Bias towards past history* π. The parameter π determines what the mean level of activation is that the activation levels are reduced to (decayed) at every timestep (activation levels are not reinitialized at every timestep). It determines in how far the 'history' of spreading activation plays a role in the action selection. This parameter can be used to tradeoff adaptivity (quick response to changes in the sensor data or goals) for bias towards the ongoing plan (inertia).

In the original algorithm presented in the previous section, these parameters have to be set by hand. There are several problems with this approach. One problem is that it is difficult to come up with the "optimal" parameter settings. It is hard to understand the characteristics of the environment and task well enough to decide what the optimal parameter values are. A second problem is that the parameters are not independent from one another, so changing the value of one also affects the others. Typically, the values for the parameter were chosen by running an experiment, noticing whether something goes wrong, tuning the parameters and starting anew. This process was iterated until the parameter values were stable. A third problem with this solution is that the parameters did not adapt to changes in the task and environment. E.g. if something about the network or the environment changed at run-time, the parameter values were not adapted to the new situation.

Therefore we decided to automate the continuous run-time adaptation of these parameters. The same agent selection algorithm is used in a "meta-level network", which runs in parallel with the network we had before. The nodes of this network do not take actions in the environment, but instead alter the parameter values for the first network so as to adapt the action selection behavior of the first network to the characteristics of the particular task and environment at hand.

The meta-level network also proved to be useful for dealing with a related problem. Occasionally we observe problems of a "non-local" nature in the action selection networks: a network can get stuck in a deadlock (the activation levels of modules are not changing anymore and none of the modules has enough energy to exceed the threshold), or in a loop (the network keeps activating the same set of modules without making progress). Noticing these problems requires a more global perspective: it is not possible for a single module to decide that the network is stuck in a deadlock or a loop. However, this proved to be easily diagnosed by the meta-level network.

The Meta-Network

The idea of the meta-net is related to other research in the Artificial Intelligence and Cognitive Science literature. Many of the AI systems that have been built have a *meta-level architecture*. They consist of two distinct levels, where one level, called the *object-level*, solves problems about and acts upon an external problem domain, while the *meta-level* solves problems about and acts upon the object-level problem solver. For an overview see (Maes & Nardi, 1988).

In our model, the meta-level control is implemented using the same action selection algorithm in a *meta-network*. The nodes (or competence modules) of this network are experts in diagnosing and curing typical problems with action selection networks. The meta network contains nine competence modules. Four of those increase and decrease the threshold and mean activation level. Three more increase the goal orientedness, data orientedness and goal conflict sensitivity. There are no meta modules that decrease these three parameters. This is the case because whenever one parameter is increased, the other parameters are decreased a little (i.e. the total sum of all the parameters is constant). We only experienced the need for increasing the sensitivity to sensor data, goals and goal conflicts. It does not seem to be necessary ever to decrease their sensitivity, except to change their respective ratios (make the action selection more goal oriented than data oriented, or more goal conflict sensitive than goal oriented, and so on). Finally there are two more meta-nodes that deal with "global" problems with the object level network: one node decreases the threshold when the object level is in a deadlock, and the other one restores the threshold to its original value afterwards. The following is a description of the different meta-nodes:

- *Make-faster* decreases the threshold θ when activated. It is triggered when the object-level network is too slow. In particular, when the average time it takes for the object network to select an action is longer than the average time it takes for the environment to change spontaneously. This is what the definition of this meta-node looks like:

```
(defmodule MAKE-FASTER
  :condition-list '(slow)
  :add-list '(appropriate-speed)
  :delete-list '(slow)
  :processes '((decrease (network-threshold
                    *object-network*))))
```

- *Make-more-thoughtful* increases the threshold θ when activated. It is triggered when the object-level is too fast, i.e. when the average time needed to select an action is shorter than the average width of the object-network (the average length of a path from executable nodes to the goal nodes). Things are slightly more complicated in that this module also takes into account how unpredictable the environment is. In a very unpredictable (dynamic) environment the threshold has to be set higher for the network to be thoughtful than in a predictable environment (one for which the object level network reliably models how it changes).
- *Make-more-biased* increases the mean-activation-level π when activated. It is triggered when the object-network is not biased enough towards the past history. This is true in two cases: (i) when the object-level "jumps" from working on one goal (selecting an action in one goal tree) to working on another goal (activating an action in a disjunct goal tree) without any good reason to do so (without the situation representing a unique opportunity, i.e. the two paths are equally long), and (ii) when the sensor data seem to be very noisy and unstable (something is true all the time and then false for just one timestep, or vice versa, something is false all the time and then true for just one timestep).
- *Make-more-adaptive* decreases the mean activation level π when activated. It is triggered when the object-level is not adaptive enough. This is the case if the object level does not react enough to important changes in the goals or the sensor data (e.g. a sudden opportunity to achieve a different goal than the one it is working on easily).
- *Make-more-goal-oriented* increases the goal-orientedness γ when activated. It is triggered when the object-level is not goal oriented enough. In particular, this is the case when, after selecting an object-level action, the total length of the tree (branching paths) from the goals to executable modules has increased.
- *Make-more-data-oriented* increases the data-orientedness ϕ when activated. It is triggered when the object-level is not data-oriented enough, i.e. when the action selection does not respond to changes in the environment fast enough, even thought the mean activation level (bias towards past history) π is low.
- *Make-more-sensitive-to-goal-conflicts* increases the goal-conflict-sensitivity δ when activated. It is triggered when the object-level is not enough sensitive to goal conflicts. This is the case when the object level is no longer able to satisfy a particular (sub-)goal (a particular proposition can theoretically not be made true anymore). This is the case when the network contains a lot of conflictor links and negative (protected) goals for which no "reversible (restoring) actions" exist.
- *Deal-with-deadlock* decreases the threshold θ when activated. It is triggered when the object-level is in a deadlock situation. This is the case when the activation levels of the object level modules change less then a small value ϵ and none of the modules has accumulated enough activation energy to exceed the threshold. Deal-with-deadlock is activated repeatedly until the threshold is low enough so that some executable module has enough activation energy.
- *Restore-threshold-after-deadlock* increases the threshold θ again after the deadlock problem has been dealt with (it restores it to its previous value). It is activated whenever deal-with-deadlock has been activated, except if deal-with-deadlock is being activated very often (in half or more of the cases that an object level module gets selected). In the latter case, the effect of deal-with-deadlock (the decrease

of the threshold) is permanent.

The meta-level network does not have any goals. It is completely data driven. Whenever the conditions for a meta-module are fulfilled (the meta-module is executable) and the meta-module has accumulated enough activation energy (from the sensor data) to exceed the meta-net threshold, the module is selected. The latter condition means that the meta-node's triggering conditions should be observed for a sufficiently long time, so that they can increase the node's activation level to a point where it exceeds the threshold. This implements the desirable feature that meta-nodes look at the average recent behavior of the object network (where the relevance of observations is weighed over time).

Discussion

The meta-level network was tested by running several experiments with very different applications. Some applications required very fast action selection, others required very thoughtful action selection, yet others emphasized goal conflict sensitivity and so on. We evaluated the system by adopting random values for the object-level parameters and then running the two networks (in parallel) until the meta-network did not select any actions anymore. Gradually the object level network displays a better and better action selection behavior, until it does not make any "wrong selections" anymore according to the meta-net's standards.

These experiments proved that the meta-net provides an effective way of setting and adapting the object-level parameters. Each time, the meta-net was able to fairly quickly settle on a selection of parameters that was satisfactory (in that no meta-level modules got triggered after a while). The meta-net does not necessarily convergence to the same parameter values for two experiments with the same application. This is the case because the same action selection behavior can be produced by "similar" parameter values (values that are within a certain interval).

The question can be raised whether there is a need for a meta-meta-net (or an infinite tower of meta-levels). In the current implementation the parameter values for the meta-network are set by hand. This is an acceptable solution because (i) the meta-net is very simple, and because of that the parameters values do not make much of a difference in its behavior, (ii) the meta-net always solves the same problem (operates in the same domain), namely that of setting the parameters of other networks, as such there is less of a need for it to adapt its action selection behavior to its environment (the environment is always the same). The only meta-level parameter which we found to be important is the threshold (and how it compares to the data-orientedness). It determines how much "evidence" has to be built up for a particular problem before the meta-net takes action and changes certain object-level parameters. It would be nice if the threshold would increase as the system proceeds, so as to implement some notion of "temperature" as in simulated annealing. This way the system would make changes to the object-level parameters more easily in the beginning and be more reluctant to change things as the object-level action selection behavior improves.

In conclusion, this experiment proves that the meta-net is a convenient method for adapting the parameters of our action selection algorithm to the characteristics of the task and environment. In addition, it proves that meta-level control does not necessarily imply a rigid, hierarchical control structure, but instead can be implemented in a distributed way. The meta-net runs in parallel with the object-net and as such does not make the latter less adaptive or reactive. Finally, more experiments will have to be performed to determine whether the current set of meta-nodes is sufficient for dealing with a wide range of tasks and environments characteristics. To this end, we plan to use "TileWorld" (Pollack & Ringuette, 1990), which is a highly parametrized environment simulator, enabling to control the characteristics of an environment.

References

AI magazine. Special issue on Universal Planning, Vol. 10 (4), 1989.

Maes, P. & Nardi, D. (editors). Meta-level Architectures and Reflection. North-Holland, 1988.

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

Maes, P. How To Do the Right Thing. Connection Science Journal. 1(3), 1989b.

Maes, P. Situated Agents can Have Goals. In: Designing Autonomous Agents: Theory and Practice from Biology to engineering and Back. P. Maes (editor). Special Issue of the Journal Robotics and Autonomous Systems. 6(1&2), 1990. Also MIT-Bradford Press book, 1991a.

Maes, P. & Brooks, R.A. Learning to Coordinate Behaviors. Proceedings of AAAI-90, 1990.

Maes, P. A Bottom-Up Mechanism for Action Selection in an Artificial Creature. In: Adaptive Behavior, from Animals to Animats. Edited by Stewart Wilson and Jean-Arcady Meijer. MIT-Press, 1991b.

Maes, P. The Agent Network Architecture (ANA). Proceedings of the AAAI Spring Symposium on Integrated Intelligent Architectures. AAAI-Press, 1991c.

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

Pollack, M. & Ringuette, M. Introducing the Tile-World: Experimentally Evaluating Agent Architectures. Proceedings of AAAI-90, 1990.