A Spreading Activation Mechanism for Decision-Theoretic Planning

Sugato Bagchi*, Gautam Biswas†, and Kazuhiko Kawamura*

*Center for Intelligent Systems
Vanderbilt University
Box 1804, Station B
Nashville, TN 37235

†Dept. of Computer Science
Vanderbilt University
Box 1679, Station B
Nashville, TN 37235

E-mail: {bagchi,biswas,kawamura}@vuse.vanderbilt.edu

1 Introduction

When performing tasks with imperfect and incomplete knowledge, actions do not always provide the expected results. Consider a fairly simple task where a robot has to pick up a spoon from a table. Many things can and do go wrong. The object being grasped may have been misclassified as a spoon. The position and orientation of the spoon may not be accurately determined by the recognition system. The table may get disturbed during the grasp operation. The spoon may be outside the robot's workspace.

Traditional planners handle these "error" situations as special cases by monitoring the environment as the pre-planned actions execute, and replanning when an unexpected state is detected [14]. The possibility of error is not considered during planning, but postponed until the execution of the plan. At the other extreme, stimulus-response systems [2] have no internal expectations of the world as they generate responses to observed stimuli. Since a predicted model of the world is not maintained, "error" situations never arise. Reactive planners attempt to combine the two approaches into one by generating new plans at execution time as a reaction to the unexpected [1, 9, 6, 4, 5].

Our interest is in reactive planners that are embedded in the task execution system and handle error situations implicitly. Given a task and a particular state of the environment the system uses decision-theoretic methods to select and execute the action most likely to achieve a state closest to the goal state. This selection is based not only upon what the effects of an action are, but also on the chances that the action will succeed in achieving those effects [4]. Consider for example, the task of picking up some food for feeding a person. A spoon or a fork can be used. When there is uncertainty associated with the food pickup operation, the planner should consider the chance of success when using the spoon or the fork, and make a choice on the basis of this knowledge. The planner described in this paper selects actions based on their *expected utility* (defined as the product of the action's desirability and its probability of success, given the existing state of the domain) in achieving the desirable effects [12]. As this action selection process continues, the goals of the task are achieved and thereafter, maintained.

If an action does not produce the expected environment (the promise of which made the action desirable), the action with the next highest utility is attempted. This may correspond to an alternate plan which was considered less likely to succeed. When such alternatives are absent (or have a very low chance of working) the same action will be retried. This process of action selection not only handles errors, but also serendipitous situations, where a highly desirable action that was unexecutable till then, suddenly finds itself executable due an unexpected change in the environment.

The obvious drawback of mixing planning with execution is that backtracking is not possible. However, unlike reactive systems, the planner can anticipate and avoid potential conflicts between actions by predicting how actions affect future states.

In our framework, uncertainty in the domain is represented by extending the definition of classical STRIPS operators to have probability values associated with the preconditions and effects of actions. The planning framework that incorporates these probabilities is introduced in Section 2. Section 3 develops and provides a formal description of a planning algorithm which considers the probability of a plan's success. Examples of planning behavior are shown in Section 5.

It is also important for real-world planning algorithms to include mechanisms that combat the computational complexity inherent in search. Connectionist systems have shown how a multitude of simple processing nodes can tackle this problem in parallel [11]. The planning algorithm described here lends itself well to implementation as a connectionist system. Actions and propositions form the nodes of this network, with directed links representing the relation between them. Communication is only between neighboring nodes, through bidirectional spreading activation. Domain knowledge is represented by the strengths of the links between the nodes. Section 4 describe how the link strengths are adapted from experience.

2 Representation of the Domain

This section describes the representation of the domain knowledge necessary for planning. The domain can be represented as a set of propositions $\boldsymbol{C} = [c_1, c_2, \ldots, c_N]$. Each proposition $c_i \in \boldsymbol{C}$ is Boolean in nature, i.e., it

can be true or false. An agent has a set of actions $[a_1, a_2, \ldots, a_M]$ that it can perform.

Figure 1 shows the representation of an action and its associated propositions as nodes of a network similar to a *plan net* [3, 5]. The strength of a link w_{ij}, represents the correlation between the connected nodes i and j. For the link, between an action a_j and one of its preconditions c_i, the correlation is defined as:

$$w_{ij} = \frac{P(a_j^{\text{success}} \mid c_i = \text{true}) - P(a_j^{\text{success}} \mid c_i = \text{false})}{P(a_j^{\text{success}})}. \quad (1)$$

This correlation is 1 (−1) if the precondition is required to be true (false) for the action to execute. Intermediate values represent *soft constraints*, which are preconditions that alter the probability of an action's success, but not definitely (i.e., with a probability of 0 or 1) as the essential preconditions or hard constraints do. They represent preconditions that are not directly related to an action, but are derived from or related to an essential precondition of the action that cannot be sensed. Moreover, soft constraints also handle situations of sensor noise, where the output of the sensor is probabilistically related to a corresponding real-world proposition. An example of a soft constraint for the spoon pickup action is the boolean proposition **table-disturbed**. When this is true, the probability of the action's success is diminished, but not down to zero, because in spite of the disturbance, the spoon may not have moved.

The link strength w_{jk}, between an action a_j and one of its effect propositions c_k is defined as follows:

$$w_{jk} = \begin{cases} P(c_k = \text{true} \mid a_j^{\text{executed}}) & \text{if } a_j \rightarrow (c_k = \text{true}), \\ -P(c_k = \text{false} \mid a_j^{\text{executed}}) & \text{if } a_j \rightarrow (c_k = \text{false}). \end{cases} \quad (2)$$

These are the prior probabilities of the action a_j's success, unconditioned on the state of the environment. A negative strength denotes that the expected state of the proposition after the action executes, is false. Based on this representation, the set of actions and their preconditions and effects can be represented as a plan net. The Markov assumption applies in determining the execution of an action: the probability of an action succeeding depends only on the state in which it executes, and not on the sequence of actions that led to this state.

3 Planning Mechanism

Let the initial situation of the environment be defined by the state S_0, where all propositions in the network are assigned a true or false value. The goal (or final) state S_F is defined by a set of propositions and their desired status (true, false, or don't care). The task for the planner is to derive a sequence of actions that transforms S_0 to S_F through a series of intermediate states.

With the domain knowledge represented as a plan net, a spreading activation mechanism implements plan generation. This is done by propagating information forward from the initial state conditions and backward from the goal conditions. The utility values, associated with the goal conditions are positive (negative) for a proposition that is required to be true (false). These are propagated backward to actions they are linked to as effects.

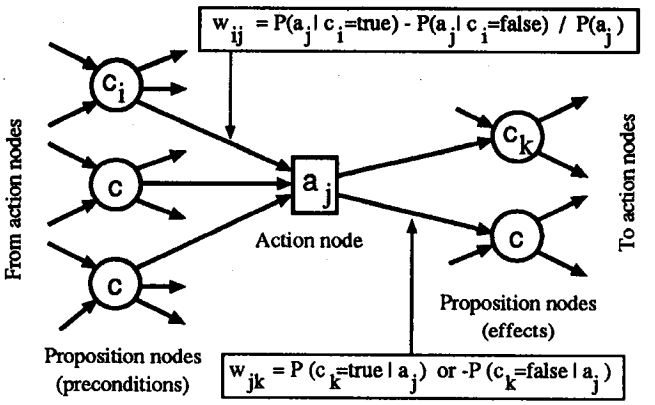

Figure 1: Representation of an action and its preconditions and effects.

If any preconditions of such actions are not in the desired state, these propositions accumulate utility values making them possible sub-goals. Back propagation can be performed iteratively from effects to preconditions through a sequence of actions.

Forward propagation is used to predict future states of the environment from its current state. The probability that a proposition will be in a given state depends on the probabilities of success of all the actions that can set the proposition to that state. This in turn affects the probabilities of success of all actions for which the proposition is a precondition. Forward propagation can be performed iteratively from preconditions to effects through a sequence of hypothetical action executions.

When the backward propagation from a desired goal condition meets the forward propagation from the initial conditions, the sequence of associated actions establish a sub-plan for achieving the goal condition. When alternate plans are found for the same goal condition, the one with the highest probability of success is chosen. This is done by accumulating in each action node, a measure of its *expected utility*, defined as the product of the probability of the action's success in achieving its effects (obtained from forward propagation) and the summed utility of its effects (obtained from backward propagation). Since the expected utility of the first action in a sequence depends on the probability of all subsequent actions succeeding, selecting the action with the highest expected utility will result in choosing the plan with the highest chance of success. The following sections describe the forward and backward propagation mechanisms in detail.

3.1 Backward Propagation of Goal Utility

The objective of backward propagation is to determine how desirable an action is, in the context of the current goals. We introduce a utility-based framework to compute the effectiveness of an action in a given situation. The utility of an action is determined from the desirability of its effects and the likelihood of goal interactions. This utility is then propagated back to the action's unfulfilled preconditions, making them sub-goals.

Consider a goal state S_F and a proposition c_k. With

respect to S_F, the utility or degree of desirability of c_k is:

$$U(c_k) = \begin{cases} > 0 & \text{if } c_k = \text{true in } S_F \\ < 0 & \text{if } c_k = \text{false in } S_F \\ = 0 & \text{if } c_k = \text{don't care in } S_F. \end{cases} \quad (3)$$

Goal conditions can be prioritized by assigning utilities of different magnitudes. c_k can also receive a utility as a sub-goal, as described later in this section.

The utility associated with c_k determines the potential reward for all actions a_j that can affect c_k. The reward for a_j due to the current state of c_k is given by:

$$R(a_j \mid c_k) = \begin{cases} w_{jk}\, U(c_k) & \text{if } w_{jk} > 0 \text{ and } c_k = \text{false} \\ & \text{or } w_{jk} < 0 \text{ and } c_k = \text{true} \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

where w_{jk} is given by Equation 2. The conditions in the above equation ensure that a reward is propagated only to actions that can change the current state of c_k. The reward will be positive for actions that promise to change the state of c_k to the desired state and negative for actions that threaten to change the state of c_k if it is already in the desired state. By this mechanism, the planner attempts to ensure that the goal state is maintained once achieved.

The expected utility of the action due to the effect c_k is defined as the product of $R(a_j \mid c_k)$ and $P(a_j^{\text{success}} \mid S_t)$, the probability of the action a_j's success under the current state S_t:

$$U(a_j \mid c_k, S_t) = P(a_j^{\text{success}} \mid S_t)\, R(a_j \mid c_k). \quad (5)$$

The method for computing $P(a_j^{\text{success}} \mid S_t)$ by forward propagation is described in Section 3.2. The overall expected utility for the action a_j, given the current state S_t and the goal state S_F, will be the summation of the utilities due to each of its effects:

$$U(a_j \mid S_t) = \sum_k U(a_j \mid c_k, S_t). \quad (6)$$

In addition to summing the utilities received from its effects, an action also propagates them to its preconditions. Only the utilities corresponding to positive rewards are propagated. This distinction is made for the following reason: Negative rewards mean that the action has undesirable effects. If a negative reward is propagated to the preconditions, it would lead to the false implication that the negation of the preconditions are desirable.

The utility received by a precondition c_i from a_j is given by:

$$U(c_i \mid a_j, S_t) = w_{ij}\, U^+(a_j \mid S_t) \quad (7)$$

where w_{ij} is given by Equation 1 and $U^+(a_j \mid S_t)$ represents the positive component of the overall utility received by the action a_j. The overall utility of the proposition c_i given the current state S_t and the goal state S_F is the summation of utilities due to each of the actions that require c_i as their precondition:

$$U(c_i \mid S_t) = \sum_j U^+(c_i \mid a_j, S_t). \quad (8)$$

A non-zero value for the above measure makes c_i a sub-goal; a positive value denotes that c_i should be true, a negative value denotes that it should be false. c_i then propagates its utility back to actions that can affect it. In addition to creating sub-goals, backward propagation also orders the actions to resolve conflicts. Negative reward is propagated to an action whose effect violates an achieved goal or sub-goal, thus inhibiting it from firing at that point in the plan sequence. This backward propagation mechanism corresponds to the "working backward from goal" control structure adopted by a number of planners (e.g. [13]). This alone is enough to generate plans. However, we also wish to consider the possibility of actions succeeding or failing from a given state. This is generated through forward propagation.

3.2 Forward Propagation of Probability

Forward propagation determines the probability of an action's success. This is determined from the state of the action's preconditions. Let S_t be the current state of the environment. This assigns true or false states to an action a_j's preconditions $[c_1, c_2, \ldots, c_N]$. We wish to know $P(a_j^{\text{success}} \mid S_t)$. Since each proposition c_i in the set can take two values, we would require to maintain 2^N conditional probabilities for each action. This can easily become memory intensive as N increases. A simpler but more approximate approach is to compute $P(a_j^{\text{success}} \mid S_t)$ from the individual conditional probabilities $P(a_j^{\text{success}} \mid c_i)$. For this, we need to make the following independence assumptions.

1. The propositions must be unconditionally independent: $P(c_1, c_2, \ldots, c_N) = \prod_{i=1}^{N} P(c_i)$.
2. The propositions must be independent given that the action executed successfully: $P(c_1, c_2, \ldots, c_N \mid a_j^{\text{success}}) = \prod_{i=1}^{N} P(c_i \mid a_j^{\text{success}})$.

Applying Bayes' rule and these simplifications, we get:

$$P(a_j^{\text{success}} \mid S_t) = P(a_j^{\text{success}}) \prod_{i=1}^{N} \frac{P(a_j^{\text{success}} \mid c_i)}{P(a_j^{\text{success}})}. \quad (9)$$

This equation forms the basis for the forward propagation mechanism. The initial value of the probability of the action a_j's success is its unconditional prior probability $P(a_j^{\text{success}})$. This is then modified by input from the action's preconditions. Each proposition c_i propagates the value $P(a_j^{\text{success}} \mid c_i)$ to the action, which "normalizes" it by dividing with the unconditional probability $P(a_j^{\text{success}})$. This *likelihood factor* will be greater than 1 if the state of the proposition c_i increases the probability of successful execution of the action and less than 1 if c_i decreases this probability. The likelihood factor, multiplied to the existing probability creates a new posterior probability. Note that input is not necessarily restricted to only the preconditions of the action. Input from propositions that are independent from the action's probability of success will have the relation $P(a_j^{\text{success}} \mid c_i) = P(a_j^{\text{success}})$, resulting in a likelihood factor of 1. If any of the actions hard preconditions are violated, for example, if c_i must be true, but is false in the state S_t, $P(a_j^{\text{success}} \mid c_i)$ will be zero, rendering the action impossible.

Equation 9 calculates the probability of an action's success under the current state of the environment S_t where the status of every proposition c_i is deterministically known to be either true or false. For planning, it is also necessary to predict future states created by the execution of actions. This is performed by forward propagation from an action to its effects. The definition of link strength w_{jk} in Equation 2 is the probability that c_k will change to a certain state (given by the sign of the strength) when a_j executes. This is unconditional upon the state of the environment during execution. To get the conditional probability of c_k in the future state S_{t+1}, obtained by executing a_j in S_t, the following computation is performed.

$$P(c_k \mid a_j^{\text{executed}}, S_t) = w_{jk} \frac{P(a_j^{\text{success}} \mid S_t)}{P(a_j^{\text{success}})} \quad (10)$$

The factor multiplied to the link strength in this equation is derived from Equation 9 as the product of the likelihood factors due to all the propositions in S_t. It will be greater than 1 if the state S_t is biased towards the successful execution of the action and less than 1 if S_t is biased against it.

Equation 10 predicts the probability of the proposition c_k from only one action a_j. Next, we consider the forward propagations received at c_k from all actions that affect it. The magnitude of each input represents the probability and the sign determines whether the action will set it true or false. The maximum input corresponds to the action most likely to set the proposition true and the minimum (negative) input corresponds to the action most likely to set it false:

$$P_{\text{best}}(c_k = \text{true} \mid S_{t+1}) = \max_j P(c_k \mid a_j^{\text{executed}}, S_t) \quad (11)$$

$$P_{\text{best}}(c_k = \text{false} \mid S_{t+1}) = -\min_j P(c_k \mid a_j^{\text{executed}}, S_t) \quad (12)$$

The best-case probability is used because it is assumed that the planner will choose the action most likely to achieve the proposition. These values are then used to compute the *best-case* predicted probabilities of actions a_l, for which c_k is a precondition. This computation is performed by Equation 9, where (substituting a_l for a_j and c_k for c_i)

$$\begin{aligned} &P(a_l^{\text{success}} \mid c_k) = \\ &\quad P(a_l^{\text{success}} \mid c_k = \text{true}) P(c_k = \text{true} \mid S_{t+1}) + \\ &\quad P(a_l^{\text{success}} \mid c_k = \text{false}) P(c_k = \text{false} \mid S_{t+1}). \end{aligned} \quad (13)$$

Here, using Equations 11 and 12,

$$\begin{aligned} P(c_k = \text{true} \mid S_{t+1}) &= P_{\text{best}}(c_k = \text{true} \mid S_{t+1}) \text{ if } w_{kl} > 0 \\ &= 1 - P_{\text{best}}(c_k = \text{false} \mid S_{t+1}) \text{ if } w_{kl} < 0 \end{aligned}$$

and $P(c_k = \text{false} \mid S_{t+1}) = 1 - P(c_k = \text{true} \mid S_{t+1})$.

In this manner, forward propagation computes predicted probabilities of the actions' success. Two features of this process are noteworthy. First, it handles both hard and soft preconditions in a uniform manner which is also effective in eliminating contributions from any unrelated propositions that are in the precondition set. To prevent the unnecessary computations involved with receiving input from propositions that have low correlation with an action's success, a *sensitivity threshold* is defined. Forward propagation is not performed if the absolute value of the link strength as defined by Equation 1 falls below this threshold. This threshold can be adjusted to determine the desired sensitivity to soft constraints, ranging from 1 where only hard constraints will be considered, to 0 when all propositions will be considered. Second, the status of the propositions do not have to be definitely known to be true or false. Intermediate values may also be used, signifying uncertainty in the sensor about the true state of the proposition. In our spoon pickup example, the object recognition system returns a certainty measure with each classification it makes. This information gets used by forward propagation to decide which spoon to pick up. Or, for the task of picking up food, the certainty measure would be one of the factors that determine which utensil to use. This determination is done by the action selection mechanism described next.

3.3 Action Selection

Given an initial state S_0, and the goal state S_F with associated utility values for the propositions in S_F, forward and backward propagations through the plan net establishes $U(a_j \mid S_t)$, the expected utility associated with executing a_j under the state S_t. To achieve an optimal plan at any point in time in the uncertain environment, an action is selected for execution if it has the highest accumulated utility value. The expected utility may change with every iteration of the forward and backward computation. The longer one waits (i.e., the more forward and backward iterations one allows) the more informed is the selection function based on the utility measure. To prevent premature executions and to give sufficient time for propagation, the planning algorithm performs the following:

1. Sum the expected utility from each propagation to create an accumulated utility for each action.
2. During each iteration, subtract a decay value D from the accumulated utility.
3. Execute an action when it satisfies:

$$\max_j \sum_t U(a_j \mid S_t), \forall a_j \left(\sum_t U(a_j \mid S_t) > T\right), \quad (14)$$

where T is a threshold that the accumulated utility must exceed.

Raising the threshold level T will require more propagations before an action can be executed. This results in increased look-ahead at the expense of additional computation time. The decay value D is inversely proportional to the amount of risk the agent is willing to take. It ensures that no additions are made to the accumulated utility of an action unless the utility in the current iteration exceeds D. For a constant reward, this places a lower bound on the probability of the action's success. A high value for D makes the agent risk-averse, requiring other actions to be performed first in order to achieve a higher probability of success for the desired action. A low value allows the agent to take more risks. A side-effect of this is that the agent will be willing to take more risks if the reward for executing the action is increased.

Note that the planner does not necessarily select the shortest action sequence, but only the sequence that has the highest probability of success. We often need to bias the optimization towards selecting the shortest sequence (necessary, for example, if all the probabilities were 1 in a domain which is no longer uncertain), or using actions that consume the least amount of resources (for example, time). For this purpose, a cost or efficiency factor η ($0 < \eta < 1$) is multiplied whenever a propagation is made through a link. If this is constant for all actions, it will favor choosing the plan with the least number of actions. Alternatively, the efficiency factor can be made inversely proportional to the cost of the resources consumed while performing each action.

4 Adaptation of Link Strengths

The strengths of the links from preconditions to actions and those from actions to effects represent the knowledge necessary for the planner described in Section 3 to operate. So far, we have assumed they are known. In this section, we look at how the strengths are learned and adapted for the domain in which the agent operates.

4.1 Identifying Preconditions

Consider a proposition c_i as a precondition for the action a_j. As the agent performs this action a number of times, either by obeying instructions from an external expert (learning by being told) or as a result of internal planning, the following statistics can be kept:

	a_j^{success}	a_j^{failure}
c_i = true	n	o
c_i = false	p	q

n, o, p, and q represent the number of times the action was performed under the different circumstances mentioned in the table. These numbers provide an estimate of the likelihood factor used in Equations 9 and 13:

$$\frac{P(a_j^{\text{success}} \mid c_i = \text{true})}{P(a_j^{\text{success}})} = \frac{n^2 + no + np + nq}{n^2 + no + np + op}. \quad (15)$$

$$\frac{P(a_j^{\text{success}} \mid c_i = \text{false})}{P(a_j^{\text{success}})} = \frac{p^2 + qp + np + op}{p^2 + qp + np + nq}. \quad (16)$$

For propositions that do not affect the success or failure of an action, $nq \simeq op$, resulting in a likelihood factor of 1. The correlation between (c_i = true) and the success of a_j, given by w_{ij} in Equation 1, can be obtained by subtracting Equation 16 from Equation 15.

4.2 Identifying Effects

The link strength between an action a_j and its effect c_k is defined in Equation 2. This can be implemented by using the following algorithm, similar to Hebbian learning:

1. Maintain the strength of the link between a_j and c_k as $w_{jk} = N_{\text{correlations}}/N_{\text{executions}}$.
2. Increment $N_{\text{executions}}$ every time the action a_j executes.
3. Increment $N_{\text{correlations}}$ whenever c_k changes to true from false after a_j executes.
4. Decrement $N_{\text{correlations}}$ whenever c_k changes to false from true after a_j executes.
5. Keep $N_{\text{correlations}}$ unchanged when the proposition remains unchanged after a_j executes.

This makes the assumption that all the effects of an action can be sensed between the time the action executes and the next. This is true for most robotic actions involving manipulating objects.

5 Examples

This section demonstrates planning behavior with regard to (a) optimal action selection, (b) error handling, and (c) goal interactions. Examples are taken from a dining table domain relevant to our ongoing research on ISAC, a robotic aid for feeding the physically challenged [7, 8]. The presence of an intelligent user allows the planner to learn the preconditions and effects of the various actions by obeying user orders and observing the consequences. Uncertainty is introduced in this domain from two sources: the incomplete knowledge present while the planner is learning from the user and the unstructured nature of the environment which is very different from a typical workcell for industrial robots. Other agents such as humans are free to modify the environment at any time, altering the state of propositions. Goal states may also be modified at will by the user.

5.1 Optimal Action Selection

For optimal action selection, the probabilities of success for the actions in a domain are varied to generate different plans, all attempting to maximize the success in achieving the goal. Consider the task of feeding solid food such as rice. Alternate plans involve using a spoon or a fork. The choice depends on two uncertainties: the probability of picking up the rice with the chosen utensil and the probability of grasping that utensil successfully.

Figure 2(a) shows the relevant actions and propositions. Note the soft constraints, grasped-spoon and grasped-fork, for the pickup-rice action. These represent the difference in the probability of succeeding in picking up the rice with a spoon or a fork. If soft constraints were not supported, two different actions would have been necessary to represent the same information: pickup-rice-with-spoon and pickup-rice-with-fork. Thus, soft constraints allow a more compact representation with fewer action nodes.

In this example, the action pickup-rice succeeded 8 times out the 12 times it was performed; 5 out of the 6 times a spoon was used and 3 out of the 6 times a fork was used. This gives an unconditional probability of 0.67 for the action succeeding. Using Equation 1, the correlation between grasped-spoon and the action is 0.5 and that between grasped-fork and the action is -0.5. The probability of the utensil grasp operations are the same (0.9). The plots in Figure 2(b) show the accumulation of expected utilities for each action. Here, the goal condition is the proposition fed-rice. The initial conditions have only the propositions located-spoon, located-fork, and located-rice true. The plan involving feeding with

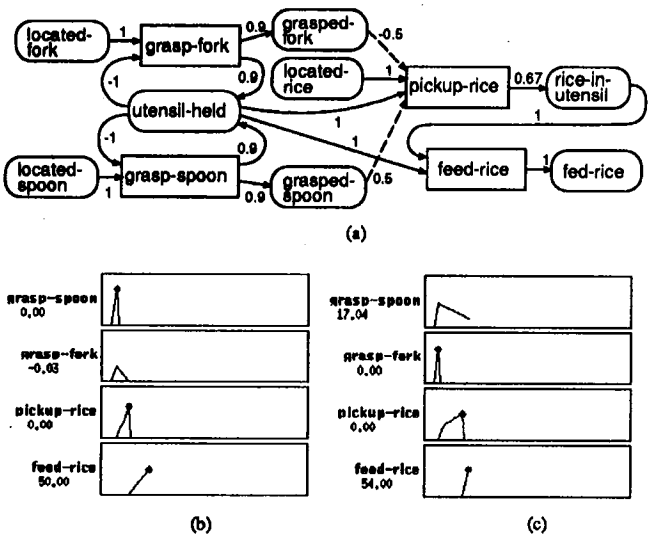

Figure 2: Optimal planning behavior in the feeding domain shown in (a). Action executions are denoted by a * symbol in the traces of the accumulated expected utility.

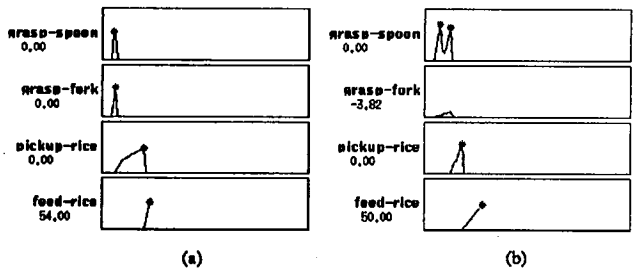

Figure 3: Two different error handling behaviors brought about by different link strengths.

a spoon was chosen because of its higher probability of success.

Figure 2(c) shows a different planning behavior where a fork is grasped for feeding, under the same initial and goal conditions. This happened when the probability of success for the action grasp-spoon was reduced from 0.9 to 0.4. Under these conditions, using the fork is a better approach because it has a higher overall probability of success when considering both the grasp and pickup operations. The same behavior is also exhibited when the object recognition system asserts spoon-located with a certainty lower than 1, as assumed so far. In such a case of uncertainty, using the fork (if recognized with a certainty of 1) has a higher chance of succeeding.

5.2 Error Handling

One of the advantages of this planner is that error handling does not have to be performed as a special case. When an action fails to make a predicted change to a condition, the planner has two alternatives. An alternate plan can be invoked, or the failed action can be retried. The choice made depends on the accumulated utilities in the competing actions. Figure 3 shows the

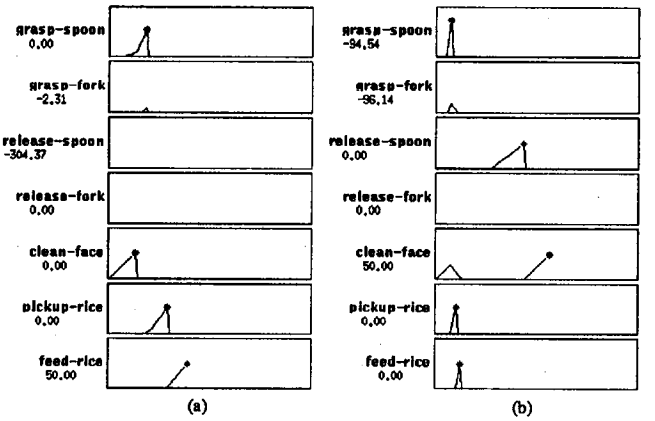

Figure 4: Planning behavior for the goals fed-rice and face-cleaned.

planner use both options for the rice feeding domain shown in Figure 2(a). When the probabilities of the actions grasp-spoon and grasp-fork are both 0.9, the planner tries grasp-fork after grasp-spoon fails, as shown in Figure 3(a). When the probability of the action grasp-fork is decreased to 0.4, the planner exhibits a different behavior, retrying grasp-spoon again because the alternate plan has an even lower chance of succeeding.

5.3 Goal Conflicts

An important drawback of many reactive systems is their inability to predict future conflicts for compound goals. Nonlinear planners handle this by explicitly looking for actions that clobber unrelated sub-goals, and then reordering the actions to avoid such clobbering. The backward propagation of utility performs this ordering in our planner. A sub-goal that is threatened by an action sends negative reward to the action thereby reducing its utility.

As an example, consider that in addition to the actions shown in Figure 2(a), the feeding robot can also perform the action clean-face. The preconditions for this action are ¬holding-utensil and located-napkin. If the proposition located-napkin and the goal face-cleaned are added to the initial conditions and goals of the examples in Section 5.1, there is potential for sub-goal clobbering. If fed-rice is attempted first, the gripper will no longer be empty. This will require an additional action release-spoon before the goal face-cleaned can be attempted. However, as shown in Figure 4(a), by ordering the action clean-face before grasp-spoon, sub-goal clobbering is avoided. This ordering is the result of the sub-goal holding-utensil sending negative reward to the action grasp-spoon.

In the above example, the utilities of both goals were 10. Figure 4(b) shows a situation where clobbering *does* take place requiring the additional release-spoon action. This happens when the goal fed-rice is assigned a higher utility of 30. The higher priority of this goal makes the planner achieve it first, unmindful of the resulting clobbering. This shows the flexibility with which planning behavior can be changed. The possibility of clobbering is

only one of the many factors considered by the planner. The other factors are goal priorities and the probabilities of success of the competing actions.

6 Conclusions and Discussion

A number of reactive planner architectures have been designed for uncertain domains. The motivation for the spreading activation mechanism for action selection was obtained from Maes' research [9]. She also uses a network for representing domain knowledge, but it is composed only of action nodes, with the propositions remaining implicit in the links between the action nodes. This results in a larger number of links than in the representation described here. For example, if there are m actions that can set a proposition and if this is a pre-condition to n other actions, Maes' representation requires mn links while our representation needs $m + n$ links. Fewer links make it simpler to adapt the link strengths. For example, if the probability of an action changing a condition has to be modified, only one link strength has to be adjusted, as compared to n in Maes' architecture.

Maximizing the probability of goal satisfaction has been studied by Drummond et al. [4]. They use a projection algorithm to explore hypothetical situations created by applying operators enabled in the current situation. This forward search uses a heuristic that estimates the effort required from a given situation to achieve the required goal. Instead of a heuristic, our approach uses backward search to come up with the utility of future actions and situations. This also allows goal interactions to be detected.

Godefroid et al. [5] describe a nonlinear reactive planner which uses forward search. It reduces the number of future situations to be explored by exploiting the independence between actions whenever possible. Knowledge about independent and dependent actions also help in the formation of nonlinear plans. However, the authors do not discuss the issue of optimization when multiple plans are available. Also, the architecture does not incorporate probabilistic knowledge which is important in uncertain domains.

Unlike traditional planners that consider only hard, deterministic relations between actions and conditions, this planner is more reliable in generating plans in uncertain domains. The action selection mechanism described here can adapt to a range of behaviors:

- From using only deterministic relations to considering relations that are uncertain to varying degrees. This is achieved by adjusting the sensitivity threshold which determines the propositions to be considered part of an action's preconditions.
- The amount of risk the planner can take may be varied. The decay value D associated with each action node is inversely proportional to the risk. If the decay is made proportional to the cost of performing the action, the planner will ensure that an action cannot be executed unless the expected utility exceeds the cost.
- The degree of look-ahead during planning can be increased by increasing the threshold T that the accumulated utility in each action node must exceed. A low value will result in a behavior similar to reactive systems. A higher value will allow the planner to consider more long term effects of actions, but at the expense of additional time.

Future tasks include studying the effect of the independence assumption between propositions. This is a major weakness of the current algorithm. Work on probabilistic reasoning[10] could be used to account for relations between propositions. Additional tests on the learning mechanism also need to be performed.

References

[1] P. E. Agre and D. Chapman. What are plans for? *Robotics and Autonomous Systems*, 6:17–34, 1990.

[2] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[3] M. Drummond. Situated control rules. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–113. Morgan Kaufmann Publishers, 1989.

[4] M. Drummond and J. Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 138–144. AAAI Press/The MIT Press, 1990.

[5] P. Godefroid and F. Kabanza. An efficient reactive planner for synthesizing reactive plans. In *Proceedings Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 640–645. AAAI Press/The MIT Press, 1991.

[6] L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. *Robotics and Autonomous Systems*, 6:35–48, 1990.

[7] A. Kara, K. Kawamura, S. Bagchi, and M. El-Gamal. Reflex control of a robotic aid system for the physically disabled. *IEEE Control Systems Magazine*, 12(3):71–77, June 1992.

[8] K. Kawamura, S. Bagchi, M. Iskarous, R. Pack, and A. Saad. An intelligent robotic aid system for human services. In *Proceedings of AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space*, 1994. In press.

[9] P. Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6:49–70, 1990. Also in Designing Automomous Agents, ed. P. Maes, MIT/Elsevier 1991.

[10] J. Pearl. *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[11] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing.* The MIT Press, Cambridge, MA, 1986.

[12] L. J. Savage. *The Foundations of Statistics.* Wiley, New York, 1954.

[13] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the IJCAI*, pages 1039–1046, 1987.

[14] D. E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.