Abstract—As robotics and automation applications extend to
the service sector, researchers have to increasingly deal with per-
forming robotic actions in uncertain and unstructured environ-
ments. A traditional solution to this problem models uncertainty
about the effects of actions by probabilities conditioned on the state
of the environment, making it possible to select plans that have
the highest probability of success in a given situation. Reactive sys-
tems use another approach to handling uncertainty, by employing
a set of predefined situation-response rules that make it possible
to move toward the goal from any situation, whether expected or
unexpected. This paper describes a planner that combines the two
approaches. A proactive component generates plans that are biased
toward picking the most reliable action in a given situation, and a
reactive component can alter the selected actions based on un-
expected situations that may arise in uncertain environments. Ac-
tion selection is driven by a spreading activation mechanism on a
probabilistic network that encodes the domain knowledge. A de-
cision-theoretic framework incorporates quantitative goal utilities
and action costs into the action selection mechanism. Experiments
conducted demonstrate the ability of the planner to plan with hard
and soft domain constraints and action costs, modify plans as a re-
action to unexpected changes in the environment or goal utilities,
and plan in situations with multiple conflicting goals.

Index Terms—Author, please supply index terms. E-mail key-
words@ieee.org for info.

I. INTRODUCTION

MOST realistic environments in which agents such
as robots operate are characterized by uncertainty.
Uncertainty can be attributed to:

1) the agent’s lack complete knowledge about the effects of
its actions;
2) uncertainty in the environment because of unexpected
changes due to the actions of other agents;
3) inability or lack of precision in the agent’s sensors to
detect certain situations, resulting in differences between
the actual and perceived state of the world.

Planning in an uncertain environment can benefit from two basic
features. The first is the ability to use past experience in deter-
mining the reliability of actions sequences. The second is the
ability to react to the unexpected changes that occur when exe-
cuting actions.

Consider the behavior of an agent that feeds a person from a
plate with food in it. The agent knows it can use either a spoon
or a fork to feed the person, but past experience has shown that
it is more reliable to use a fork than a spoon for this kind of
food (e.g., spaghetti). However, at present, only a spoon can be
found. The agent generates a plan for feeding with the spoon and
picks it up. At this point, it suddenly detects a fork. Rather than
continue to feed with the spoon, the agent modifies its plans,
puts the spoon down, picks up the fork, and proceeds to feed the
person with the fork.

This behavior though seemingly simple, cannot be achieved
by most planners, even those that are designed to handle uncer-
tainty. Classical planners, surveyed in [1], assume that the ef-
fects of an action are known with certainty. With this knowledge,
they generate a set of actions that will achieve the desired goals.
Since the reliability of actions are not modeled, the planner can
only decide whether plans are implementable or not, but have no
way to reason about the chances of success of competing plans.
Some planners do monitor for errors as actions are executed [2].
An error is signaled if the observed state of the world does not
match the expected state after the execution of an action. How-
ever, unexpected situations are not always synonymous with ex-
ecution errors. In the example above, the detection of the fork
was unexpected, but that does not imply that there was an error.
Yet, it rightfully triggered a change in plans.

Some planners assign probabilities to the effects of actions
[3]–[5]. This allows comparison of the reliability of competing
plans, but provides no mechanisms for reacting to unexpected
changes in the environment. Reactive planners [6]–[8] are de-
signed to select and execute actions in response to the current
state of the world, thus interleaving planning and execution.
This makes them more responsive to unexpected changes. How-
ever, with a few exceptions [9], [10], they do not use proba-
bilistic information to determine the likelihood of success of the
actions. They may be good at handling the unexpected, but they
do not attempt to alleviate the uncertainty of future success by
choosing more reliable actions.

The inability of existing planning systems to reliably deal
with uncertainty arising from unexpected changes in the envi-
ronments, like the feeding scenario described above, motivates
the research described in this paper (see also [11] and [12]). The
objective is to incorporate a reactive planner’s ability to handle
unexpected situations with a probabilistic planner’s proactive
ability to select the most reliable plan (or plan step) in a given
situation. Goals that the agent must achieve are assigned numerical
utilities to indicate their importance. This representation allows
the selection of actions on the basis of their expected utilities,
which is the product of an action’s utility and the probability of

Task Planning under Uncertainty using a Spreading
Activation Network

Sugato Bagchi, Gautam Biswas, Senior Member, IEEE, and Kazuhiko Kawamura
its success in achieving those effects. Changes in the state of the environment or the utility of goals affect the expected utilities, allowing the system to react to these situations and if necessary, to dynamically revise plans.

II. PROBABILISTIC REPRESENTATION OF UNCERTAINTY FOR PLANNING

Consider a robot assigned the task of using a fork to pick up french fries from a plate. To be completely certain about the success of the action, the robot needs sensors to detect relevant conditions, such as the exact position of the plate and fork, the position and orientation of the individual fries so that the robot may determine which one to pick next and how, the hardness of the fries to determine the proper force to apply, and the sharpness of the fork. It is unlikely that all these details will be sensed directly or computed with any precision because of the expense and complexity of the sensors and computational mechanisms required. Performing the pick up action without all this information implies some uncertainty about its success.

A condition that is crucial for the success of an action may not be sensed by the agent, because it is not aware of its significance or because the condition is impractical to sense. But the probability of success of the action depends directly on the unsensed condition. Sometimes, a dependency between the condition and some other sensed condition, expressed as a probabilistic relation can be exploited. For example, knowing that the plate is full of fries would imply a high probability that the robot could position the fork on a part of the plate with fries in the right orientation for a successful pickup. Since this relation is not completely certain, we define plate-full as a soft constraint for the pickup-fries action. If this condition is true, the action is more likely to succeed than if it is false. For sensed conditions, sensor noise can be the cause of uncertainty. For example, suppose a camera is used to detect the fork’s position on the table. Errors in the position data can occur due to improper segmentation of the image (perhaps due to bad lighting, shadows, and overlapping objects) and inaccurate transformation from camera coordinates to robot coordinates (perhaps due to a slight movement of the camera, resulting in improper calibration).

To generalize, there are two particular situations that give rise to uncertainty in performing an action:

1) imperceptible conditions, where an actual precondition of an action is not sensed and cannot be derived from the other conditions,
2) inferred conditions, where the actual precondition is not sensed directly but it is probabilistically related to conditions which are. This includes the case of sensor noise.

We describe a representation of planning knowledge that incorporates this uncertainty in the form of a directed graph containing propositions describing situations in the environment, and actions that can change these situations.

A. Representation of Actions for Planning Under Uncertainty

In our work, we assume that the domain is described by a set of Boolean propositions \( C = \{ c_1, c_2, \ldots, c_N \} \), \( \forall c_i \in C, c_i = \text{true} \lor c_i = \text{false} \) that it can perform. The representation for actions corresponds to the traditional STRIPS formalism [13] with three parts: 1) a precondition formula; 2) an add list; and 3) a delete list. The precondition formula must be true in the current state before the action can be executed successfully. When executed, the new state is obtained by deleting all propositions specified by the delete list and adding the propositions appearing in the add list. The add and delete list can be combined into a single effects list (as in our representation) where propositions whose states change value (between true and false) are explicitly recorded.

Our representation is adapted to include probabilistic information about the domain. Fig. 1 shows the graphical representation of an action and its associated propositions as nodes. Proposition nodes with links pointing to an action node are its set of conjunctive preconditions. The strength of a link \( w_{ij} \), represents the correlation between the connected nodes \( i \) and \( j \). Proposition nodes with links pointing to them denote the effects of the actions where the links originate. The strength of such a link, \( w_{jk} \), represents the probability with which the action achieves this effect.

For the link between an action \( a_{ij} \) and one of its preconditions \( c_i \), the correlation is defined as

\[
\begin{align*}
\alpha_{ij} = P(a_{ij} = \text{true} | c_i = \text{true}) - P(a_{ij} = \text{false} | c_i = \text{false}) \\
\alpha_{ij} = P(a_{ij} = \text{true} | c_i = \text{false}) + P(a_{ij} = \text{false} | c_i = \text{true})
\end{align*}
\]

(1)

\( w_{ij} \in [-1, 1] \) defines the correlation between the success of an action and its preconditions. When its value is 1 (−1), the precondition is required to be true (false) for the action to execute, i.e., it represents a hard constraint for an action. Preconditions with intermediate correlation values represent soft constraints. They increase or decrease the probability of an action’s success. If \( w_{ij} \) is positive, the probability of \( a_{ij} \’s \) success is higher if \( c_i \) is true. If \( w_{ij} \) is negative, the probability of success increases if \( c_i \) is false. As discussed earlier, soft constraints can also model sensor noise, where the noisy output of the sensor is probabilistically related to a corresponding real-world proposition.
The link strength $w_{jk}$, between an action $a_j$ and one of its effect propositions $c_k$, is defined as follows:

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

(2)

$w_{jk}$, which ranges from $-1$ to $1$, is based on the prior probabilities of the action $a_j$'s success, unconditioned on the state of the environment. A negative strength denotes that the proposition becomes false after the action executes.

This representation defines a directed network from preconditions through actions to their effects. Plans can be represented as paths through this network, linking a sequence of actions to achieve one or more goals. Actions along the path necessarily satisfy preconditions for the following actions, and this ultimately leads to the achievement of goal propositions. Section III describes a spreading activation mechanism for creating such paths between the observed propositions in the current state and the desired propositions in the goal state.

The probability of success of actions is conditioned on individual propositions rather than the state of the environment. This “proposition-space” representation reduces space and learning complexity when compared to “state-space” representations: actions form relations with individual propositions rather than the exponentially larger set of states. As a result, the task of learning the conditional probabilities are also simplified. The drawback is that the dependencies among propositions have to be explicitly modeled and handled during the computation of probabilities. State-space approaches, such as Markov decision processes, avoid the independence assumption since the probabilities associated with each state are considered “atomic,” and not synthesized from the probabilities of the constituent propositions.

Another distinctive feature of this representation is the explicit modeling of both actions and propositions. Many of the previous network representations make either the actions or the propositions implicit. Bayesian networks represent propositions as nodes, while actions are implicit in the links between propositions [14], [15]. This is not appropriate for generating plans as a sequence of actions. The spreading activation representation used by Maes [8], [16] 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 certain proposition which is a pre-condition to $n$ other actions, Maes’ representation requires $m\cdot n$ 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.

III. A Spreading Activation Mechanism for Action Selection

Unlike classical planners, the spreading activation mechanism for action selection does not create a complete plan. Instead, as Fig. 2 illustrates, it operates like a feedback control system, choosing and executing selected actions one at a time as long as there is a difference between the current state and the goal state. This approach addresses the dual objectives of goal achievement and goal maintenance.

Let the current state of the environment be represented by a set of boolean propositions $S$. A goal $G$ is defined as a subset of propositions that must achieve desired values (true or false) for a task to be successful. The objective of the system is to reduce the error between $S$ and any state $S_f \supseteq G$. This is done by propagating information forward from the current conditions (the current state of the world) and backward from the goal propositions (the desired state of the world). A goal proposition is given a numeric utility value that denotes how desirable it is for the proposition to achieve the specified value. These utilities are propagated backward (opposite to the link direction) to actions which can change the value of the proposition. If any preconditions of these actions are not at their desired values, these propositions accumulate utility making them possible sub-goals. Back propagation can be performed iteratively from effects to preconditions of actions, thus creating a sequence of sub-goals (not unlike STRIPS-like planners).

Forward propagation predicts the probability of future states of the environment given its current state. The probability that a proposition will assume a given value depends on the probabilities of success of all the actions whose execution can set the proposition to that state. This in turn affects the probabilities of success of all actions that have this proposition as a pre-condition. Forward propagation is performed iteratively from preconditions to effects through a sequence of hypothetical action executions.

When the backward propagation of utility from a desired goal proposition meets the forward propagation of proposition probabilities from the initial state of the environment, the sequence of associated actions establish a sub-plan for achieving that goal. When alternate plans are found for the same goal proposition, the one with the highest probability of success is chosen. This is done by calculating for 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 utility of its effects (obtained from backward propagation) [17]. Since the expected utility of the first action in a sequence depends on the expected utility of all subsequent actions, selecting the action with the highest expected utility is equivalent to choosing the plan with the highest expected utility. If no changes are made to the desired goal state and the state of the environment changes only as expected due to the execution of the actions, following the sequence of actions with highest expected utility produces, in some sense, the “optimal” plan. If unexpected events do occur, choosing the action with the highest expected utility is equivalent to following a plan that is most likely to succeed while optimizing the cost of the actions in the plan. This issue is discussed later in this section.

The forward and backward propagation mechanisms are described in greater detail next.

The word “value” is not to be confused with “utility.” Throughout this paper, “value” of a proposition means its status (true or false). “Utility” expresses the desirability that a proposition assume a certain value.
A. Backward Propagation of Goal Utility

The objective of backward propagation is to determine the desirability of actions in the context of the current goals. We introduce a utility-based framework to assign values that correspond to the desirability of propositions in the goal set \( G \). With respect to \( G \), the utility or degree of desirability of any proposition \( c_k \) is

\[
U(c_k) = \begin{cases} 
0 & \text{if } (c_k = \text{true}) \in G \\
<0 & \text{if } (c_k = \text{false}) \in G \\
0 & \text{if } c_k \notin G.
\end{cases}
\]  

(3)

The priority of goal conditions is set by assigning utilities of different magnitudes. \( c_k \) may also be a sub-goal, whose utility value is derived from the goal utility, as described later in this section.

The utility associated with \( c_k \) determines the potential reward for all actions \( a_j \) that can change \( c_k \) from its current value in state \( S_t \) to the desired value in \( G \). The uncertainty in the current value of \( c_k \) is expressed as a probability, \( P(c_k = \text{true}|S_t) \). \( P(c_k = \text{true}|S_t) = 1 \) implies \( c_k \) is definitely true, and actions that set \( c_k \) to true should not receive any reward since they are redundant. Similarly, if \( P(c_k = \text{true}|S_t) = 0 \), actions that set \( c_k \) to false should not get a reward. These restrictions are ensured by the following equations for assigning reward to an action \( a_j \).

\[
R(a_j|c_k, S_t) = \begin{cases} 
\eta w_{jk} P(c_k = \text{false}|S_t) U(c_k) & \text{if } (w_{jk} > 0) \\
\eta w_{jk} P(c_k = \text{true}|S_t) U(c_k) & \text{if } (w_{jk} < 0)
\end{cases}
\]

(4)

where \( w_{jk} \) given by (2), is positive (negative) if \( a_j \) sets \( c_k \) to true (false). \( 0 \leq \eta \leq 1 \) is an attenuation factor that biases action selection toward the shorter of two plans with equal expected utilities.\(^2\) The reward will be positive for actions that promise to change \( c_k \) to the desired value, and negative for actions that threaten to change \( c_k \) if it is already at the desired value. This mechanism allows the planner 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

\[
U(a_j|c_k, S_t) = P(a_j|\text{success}|S_t) R(a_j|c_k, S_t).
\]

(5)

The method for computing \( P(a_j|\text{success}|S_t) \), the probability of actions \( a_j \)’s success under the current state, \( S_t \), by forward propagation is described in the next section. The overall expected utility for the action \( a_j \), given the current state \( S_t \) and the goals \( G \), will be the summation of the utilities due to each of its effects. From this, the cost \( C(a_j) \) of performing the action is subtracted. This forms the basis for a decision-theoretic approach, where the expected utility of a decision is compared with its cost, and the “profit” for taking that decision is computed as \([18]\)

\[
U(a_j|S_t) = -C(a_j) + \sum_k U(a_j|c_k, S_t).
\]

(6)

The cost may represent resources consumed by the action when it is executed. This allows for the optimization of resource consumption along with expected utility.

\(^2\)This will be the case in deterministic domains.

In addition to summing the utilities received from its effects, an action also propagates the utilities corresponding to positive rewards, \( U^+(a_j|S_t) \) to its preconditions. Negative rewards, which imply that the action has undesirable effects, are not propagated. Their propagation to preconditions would lead to the false conclusion that the negation of the preconditions are desirable.

The utility received by a precondition \( c_j \) from \( a_j \) is given by

\[
U(c_j|a_j, S_t) = \eta w_{ij} U^+(a_j|S_t)
\]

(7)

where \( w_{ij} \) is given by (1). The overall utility of the proposition \( c_j \) is the summation of all the utilities it receives from each of the actions that require \( c_j \) as their precondition

\[
U(c_j|S_t) = \sum_j U(c_j|a_j, S_t).
\]

(8)

A nonzero value for the above measure makes \( c_j \) a sub-goal. A positive value for this measure denotes that \( c_j \) should be true and a negative value denotes that it should be false. \( c_j \) then propagates its utility back to actions that can affect it [\([4]\)]. In addition to creating sub-goals, backward propagation also helps to order actions and 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 goal regression control structure adopted by a number of planners [\([13]\), \([19]\), \([20]\)]. It is clear that this mechanism by itself is enough to generate plans. However, we wish to guide the search for a plan while accommodating the possibility that actions may succeed or fail when applied in a given state. The success of actions is probabilistically determined by forward propagation.

B. Forward Propagation of Probability

Forward propagation determines the probability of an action’s success conditioned on the state of the environment. Let \( S_t \) be the current state of the environment, defined by assigning a true or false value\(^3\) to all propositions \( [c_1, c_2, \ldots, c_N] \). In general, computing \( P(\alpha^\text{success}|S_t) \) determining \( 2^N \) conditional probabilities for each action, which is both computation and memory intensive for large \( N \). A simpler approximation computes \( P(\alpha^\text{success}|S_t) \) assuming the conditions are independent of each other:

1) \( P(c_1, c_2, \ldots, c_N) = \prod_{i=1}^{N} P(c_i) \),
2) \( P(c_1, c_2, \ldots, c_N|\alpha^\text{success}) = \prod_{i=1}^{N} P(c_i|\alpha^\text{success}) \).

Applying Bayes rule along with these simplifications, we get

\[
P(\alpha^\text{success}|S_t) = P(\alpha^\text{success}) \prod_{i=1}^{N} \frac{P(c_i|\alpha^\text{success})}{P(c_i)}. \]

(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 \( P(\alpha^\text{success}) \). This is then multiplied by the input from the action’s preconditions, \( P(c_i|\alpha^\text{success}) \).

In reality, because of uncertainties in sensing and computation, the value of a proposition may be uncertain, i.e., \( 0 \leq \begin{cases} 
\end{cases} \]

\(^3\)We will consider probabilistic knowledge of proposition values later.
This likelihood factor will be greater than 1 if the value of the proposition $c_i$ increases the probability of successful execution of the action, and less than 1 if $c_i$ decreases the probability. Propositions unrelated to the success of the action will satisfy the following relations:

$$P(a_j^{\text{success}} | c_i = \text{true}) = P(a_j^{\text{success}} | c_i = \text{false}) = P(a_j^{\text{success}}),$$

(11)

This makes the likelihood ratio 1 irrespective of what $P(c_i = \text{true}|S_t)$ is. If any of the action’s hard preconditions are violated, for example, if $c_i$ must be true for the action to succeed (i.e., $P(a_j^{\text{success}} | c_i = \text{false}) = 0$), while $P(c_i = \text{true}|S_t) = 0$, the null propagation will make the action’s posterior probability equal to zero.

Equation (9) calculates the probability of an action’s success given the current state of the environment $S_t$. For planning, it is also necessary to predict future states that result from the execution of actions. This is performed by forward propagation from actions to their effects. $w_{jk}$, as defined in (2), is the probability that $c_k$ will change to a certain state when $a_j$ executes, independent of the state of the environment during execution. To get the conditional probability of $c_k$ in the future state $S_{t+k}$, obtained by executing $a_j$ in $S_t$, i.e.,

$$P(c_k | a_j^{\text{executed}}, S_{t+k}) = w_{jk} P(a_j^{\text{success}} | S_t).$$

(12)

Equation (12) predicts the probability of proposition $c_k$ with respect to action $a_j$. When multiple actions affect $c_k$ a number of probability terms are forward propagated to it. The magnitude of each input represents the probability and the sign determines whether the corresponding action will set the condition to true or false. The maximum input corresponds to the action most likely to make the proposition true and the minimum (negative) input corresponds to the action most likely to make it false

$$P_{\text{max}}(c_k = \text{true}|S_{t+k}) = \max_j P(c_k | a_j^{\text{executed}}, S_t)$$

(13)

$$P_{\text{max}}(c_k = \text{false}|S_{t+k}) = -\min_j P(c_k | a_j^{\text{executed}}, S_t).$$

(14)

Making the assumption that actions are performed sequentially, the highest probability value is propagated to proposition $c_j$ because it is assumed that the planner will choose the action most likely to achieve the desired value for the proposition. These values are then used to compute the best-case predicted probabilities of actions $a_k$ for which $c_k$ is a precondition. In this manner, forward propagation computes the probability of success of different actions, given the current state of the environment. Two features of this process should be noted:

1) A probability value is associated with the propositions. This probability signifies the uncertainty in the value of the proposition. In our food pickup example, the object recognition system may associate a probability value with each object it recognizes. The value signifies how certain the system is about the identity of the recognized object, and this is one of the factors used in making the decision whether to use a spoon or fork. This decision-making process is discussed in Section III-C.

2) Link weights differentiate essential propositions ($\{w_{jk} = 1\}$ from soft constraints ($\{w_{jk} < 1\$). The forward propagation handles both types of conditions in a uniform manner. To prevent a lot of unnecessary computations involving soft constraints with low correlation to an action’s success, a sensitivity threshold is defined. Forward propagation from a precondition to an action is not performed if the absolute value of the link strength, as defined by (1), falls below this threshold. This threshold can be adjusted to determine the degree of detail, ranging from 1 where only essential preconditions are considered, to 0 when all propositions are considered. Increasing the sensitivity threshold effectively increases the abstraction level at which planning is done [21], [22].

C. Action Selection

Given an initial state $S_0$, and the goal set $G$ with associated utility values for the propositions in $G$, forward and backward propagations through the plan net establishes $U(a_j|S_t)$, the expected utility associated with executing $a_j$ under the state $S_t$. The expected utility may change with every step of the forward and backward propagation.

The action selection mechanism simply selects the action with the highest expected utility. This selection may be done at any time, but the longer one waits (i.e., the more forward and backward propagations one allows) the more informed is the selection function based on the utility measure. Thus, the mechanism has the “anytime” property [9], [23]. Increasing the allotted planning time allows a larger number of propagation steps before selecting an action, making the planner’s behavior more deliberative. Examples presented in Section IV demonstrate this property.

Since utility originates in the goal propositions, an action will receive utility only if it is in a “path” of actions and propositions that lead to one or more goal propositions. Referring to (3)–(8), the utility received by an action represents the product of the probabilities of success of the subsequent actions in the path leading to the goal proposition. Selecting the action with the highest expected utility implicitly selects the plan (i.e., the sequence of actions along a path) with the highest expected utility. In our framework, the highest expected utility trades off reliability against action cost. A highly reliable action may not be performed because of its high cost. If reliability is of paramount importance, the cost factor can be dropped from the utility computation.

The action selection mechanism is a compromise between classical deliberative planners and reactive planners. Classical planners compute one path from an initial state to a goal state, and can only react to unexpected changes by complete
replanning. This can be very expensive. Triangle tables [13] are an alternative to replanning, but they assume that all possible combinations of conditions are made explicit and the action to take from each situation is computed a priori. This is not possible when the state of a proposition is only probabilistically known—there are infinitely many states in this case. On the other hand, reactive systems have no prior knowledge about consequences of their actions sequences, and therefore, do not generate a complete plan to achieve a goal. They do not have the computational burden of replanning, however, no guarantees can be made about the efficiency of the plans they generate. Classical planners have a better chance for selecting efficient plans because they evaluate consequences of action sequences before actual execution.

Our action selection mechanism assumes knowledge of the effects of actions but does not assume that they will be successful. Action selection is performed dynamically, one action at a time. The action selected is the one with the highest utility under the current state of the environment. The next action selection step takes into account the state of the environment after the current action has been executed. Therefore, if the previous action failed to have the desired effect, no special “error” handling routines have to be invoked. The choice of the next action may correspond to an alternate plan which was previously considered less likely to succeed, or if alternate plans do not exist (or still have a lower chance of success), the failed action may be retried again. This process of action selection not only handles errors, but also serendipitous situations, where a highly desirable action with a low probability of success suddenly has its probability increased by forward propagation from an unexpected change in the environment. Another source of unexpected situations is a change in goal propositions or their utilities. These changes are inserted into the network, altering the expected utilities of actions. When the environment or goals change, they are immediately conveyed to the nodes directly affected by them, but there may be a delay before these changes reflect on the action nodes that are competing for selection. The magnitude of this delay depends on the length of the plans being considered.

An action that is desirable in the context of one goal proposition may be undesirable in the context of another. Planners typically avoid these situations by ordering actions to avoid harmful goal interactions [24]. For example, if an effect of one desirable action \( a_i \) violates a precondition of another action \( a_j, a_j \) should be performed before \( a_i \). In this planner, such orderings are imposed in the action selection mechanism through back propagation of utility. \( a_i \) will receive negative utility from the precondition of \( a_j \) it affects. If the positive utility coming into both actions are the same, this will ensure that the utility of \( a_j \) is higher than that of \( a_i \). The assignment of utilities to goal propositions enable prioritization of actions to allow higher utility goals to be achieved first, even at the expense of sub-goal clobbering. In the previous example, if \( a_j \) receives a much higher positive utility than \( a_i, a_i \) will be performed first even if it means that some other action(s) are needed before \( a_j \) can be performed. In this way, the utilities can be used to describe the preferred sequence of goal achievement, even if this requires extra actions. Also, when the goals are contradictory, i.e., achieving one violates another, utility values are used to choose between goals. This cannot be done when goals are treated symbolically with equal priority or utility.

D. Learning the Link Strengths

In terms of the domain knowledge representation described in Section II, the objective of the learning task is to assign the correct weights to the links between preconditions and actions and between actions and their effects [(1) and (2) in Section II].

Our objective for learning action preconditions and effects from observation is similar to past work [25]–[27], but our approach of using correlational measures to obtain a probabilistic relation between actions and propositions is different. The past approaches assume the domain is deterministic. The system learns domain knowledge by observing 1) the state of the environment; 2) the action executed in that state; and 3) the effects of the action on the environment. Before learning can occur, it is assumed that the system knows how to perform actions and how to use its sensors to gain information about the current state of the environment. These procedures can be produced by the designer or taught to the system by a rote learning methodology. This procedural knowledge enables the robot to perform the individual actions. However, the execution of a task, such as feeding, requires the successful execution of a sequence of such actions. We assume that the robot behaves like a learning apprentice [28], [29], which is initially guided by the user or a teacher, but with time, learns the preconditions and effects of actions by induction. This knowledge is obtained by observing the state of the environment before and after the execution of each action. Thus, with time, the system gains enough domain knowledge to relieve the user from the burden of planning and directing the robot during task execution.

The learning approach described here makes the assumption that the effects of any action can be detected as soon as the action is completed. One does not need to wait till the goal state is reached before getting a feedback about success or failure. For the actions a robot may take, this is not an unfair assumption. As a result, it is possible to learn about individual actions as a goal is being achieved and then to apply that knowledge to other goals that may require the same action. To be statistically relevant, a large number of observations are necessary. In practice, it may not be possible to acquire a large number of randomly ordered examples. The effect of learning from a few observations that are always in a definite sequence and the steps taken to improve learning under this situation is described in [30].

E. Coping with Loops

The local propagations performed by the spreading activation mechanism are well suited for parallel computation. However,
pure local propagations cause problems when loops are present in the network. Loops occur in two forms: 1) undirected loops, where the direction of the links do not matter and 2) directed loops, where the direction of the links is honored. Only undirected loops may occur in Bayesian networks, not directed ones. The effect of these loops on the planning process is described briefly in this section. Details on the handling of loops may be found in [30].

Undirected loops are essentially multiple paths that originate from one proposition and terminate into another. The spreading activation algorithm, as described before, will perform two types of miscalculations in the presence of undirected loops. First, the utility of achieving a proposition may be summed multiple times. Second, the probabilities of actions and propositions along the multiple paths originating from a common proposition will not be independent, and should not be multiplied at the terminating node. Directed loops represent the presence of positive and negative feedback in the network. Such situations do not occur in Bayesian networks, so no mechanisms for handling them have been developed. As discussed in Section II, the absence of feedback loops cannot be assumed for planning nets. The spreading activation process will cause an unbounded increase of utility in the presence of positive feedback and oscillations in case of negative feedback.

These miscalculations are avoided by detecting the presence of loops during the spreading activation process. The algorithm is modified to avoid the summation of utilities originating from a common source (goal). The violations of the independent probability assumption are handled by detecting the common dependencies such that the common prior probabilities are considered just once in computing the dependent probabilities. This is similar to the method of conditioning performed in Bayesian networks [14]. Details on how these are performed, with a bounded increase in time and space complexity, are given in [30].

Our current representation ignores the presence of loops arising due to causal links between propositions. These represent dependencies between propositions that are not mediated by actions the agent performs, and are best represented by Bayesian networks [14]. A merger of propagation mechanisms through Bayesian and planning networks would, therefore, create a more realistic and robust plan generation. The next section presents empirical results that support the effectiveness of the planner in many of the operating situations described above.

IV. EXAMPLE OF PLANNING BEHAVIOR

This section presents experiments to demonstrate empirically, the properties and capabilities of a planner based on the action selection, such as the ability to:

1) plan with hard and soft constraints and probabilistic information as a representation of uncertainty in domain knowledge;
2) generate plans that consider goal utilities and action costs;
3) modify plans (without complete replanning) to handle unexpected changes in the environment and goal utilities; and
4) plan in situations with multiple conflicting goals.

Additional examples are described in [12].

A. A Planning Test Bed

A “test bed” has been implemented for empirically testing the planner’s behavior under various situations. The test bed incorporates all the domain knowledge that will be used by the planner. This knowledge consists of:

1) A complete list of actions that can be performed by the agent. Every action is defined by a set of preconditions and a set of effects. These sets may be incomplete. 2) Relevant conditions in the environment that can be sensed by the agent.

The description language also allows the representation of mutually exclusive propositions such as [grasped (spoon), grasped (fork), grasped (nothing)], probabilities that qualify soft constraints for actions and the achievement of an effect after an action is performed, and goal utilities and action costs. This description language allows the creation of the required plan net with links of specified strengths.

In addition to the domain knowledge, a planner also needs to be given the current state of the environment and the goals. Also, since the environment can be dynamic, i.e., proposition values and goal utilities can change at any time, some means for signaling these changes to the planner are required. This is done with a graphical user interface for the planning test bed. The user (or a software agent) can set a proposition to true or false during plan generation and execution. Similarly, goal utilities can be changed at any time. These features make it possible to test the reaction of the planner to unexpected changes in the environment and to changes in the goal priorities. The number of propagation steps to be performed before selecting the action with the highest expected utility can be set by the user to adjust the reactive or deliberative nature of the planner. Actions can also be forced to execute at any time by clicking the corresponding label. This allows the system to learn or modify the supplied domain knowledge.

1) The ISAC Environment: The primary domain for our experiments is the ISAC (Intelligent Soft Arm Control) system, a robotic aid developed at Vanderbilt University [31], [32], for feeding the disabled. Consider the task of using a robot to pick up some food from a plate for feeding a disabled user. The ISAC system uses a pneumatically actuated robot manipulator for grasping utensils, picking up food from a plate or bowl, and feeding. A camera is used for recognizing and locating common tableware, including spoons, forks, cups, bowls, and plates on the table. To ensure that the food is placed in front of the user’s mouth, a pair of cameras track the user’s face in three dimensions.

The unstructured nature of the environment is the cause for uncertainty in this domain.

1) The object recognition system can not recognize objects with absolute certainty. A probability is associated with each of the recognized objects. 2) Inaccuracies in camera calibration can lead to grasping errors. 3) Success in picking up food depends on its position in a plate or bowl, but the exact positions are not sensed.
4) People are free to modify the environment while the robot is executing a task.

5) The user is free to change his goals or bias the plan selection process by specifying sub-goals.

This domain demonstrates the need for handling unexpected change of state in the environment and unexpected changes or modifications to the goals.

### B. Probabilistic Planning

Even when the environment is static, there is uncertainty in the robot’s actions. When other factors, such as action costs and utilities are equal, the planner must choose a plan that has the highest probability of success. Based on information gathered from previous task executions, the following characteristics of the domain should influence the robot’s actions:

1) In past feeding activities, the food pickup action was successful six times out seven when a fork was used and the plate was full. When the plate was not full, the fork was successful in only three of nine attempts.

2) When a spoon was used, the action was successful in only one of five times when the spoon was small (a teaspoon) and nine of 13 attempts when the spoon was larger.

3) Grasping the fork was successful 18 times out of 20.

4) Grasping the spoon was successful 15 times out of 17.

The propositions plate-full is a soft constraint for the action use-fork which picks up the food from the plate. The essential precondition is whether there is enough food in the proper orientation at the spot on the plate where the arm will attempt a pickup with the fork. Since this is hard to sense, a related condition: whether the plate is full enough, is sensed. When this is true, it increases the likelihood of the essential condition being true. Note that the proposition plate-full need not be assigned a true or false value (as it is in this example). It could have a probability value associated with it. small-spoon is a soft constraint for the action use-spoon that reduces the chances of success of the pickup action.

What should the planning behavior be, given different initial states? Table I shows four initial states under which the planner has to decide which utensil to grasp. Assume that the necessary hard preconditions (such as located (spoon), located (fork), hand-empty, etc.) are all satisfied. The planner uses the probabilities of success in picking up the food listed in the table to make its decision. Consider the situation where plate-full is true and, therefore, the fork is being used to feed. Fig. 3(a) shows the traces of the expected utility for the relevant actions when small-spoon is false. (The x-axes of these traces represent time spent in interleaved planning and execution. The execution points are represented by “*”.) After two iterations with the fork, the proposition plate-full becomes false. Due to the change in the probability of success of using the fork, it is released and the spoon is used instead. This behavior can be explained by comparing lines 1 and 3 in Table I. Fig. 3(b) shows a different planning behavior when the condition small-spoon is true. This time, when the condition plate-full turns false after a couple of iterations, the planner continues to use the fork because the probability of success of using a small spoon is even less (see line 2 in Table I).

Fig. 3. Action selection based on the value of soft constraints. Expected utility for each action is plotted against propagation steps. Action executions are denoted by a star.

<table>
<thead>
<tr>
<th></th>
<th>small-spoon</th>
<th>plate-full</th>
<th>Probability of</th>
<th>Probability of</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>false</td>
<td>false</td>
<td>0.60</td>
<td>0.30</td>
</tr>
<tr>
<td>2</td>
<td>true</td>
<td>false</td>
<td>0.18</td>
<td>0.30</td>
</tr>
<tr>
<td>3</td>
<td>false</td>
<td>true</td>
<td>0.60</td>
<td>0.77</td>
</tr>
<tr>
<td>4</td>
<td>true</td>
<td>true</td>
<td>0.18</td>
<td>0.77</td>
</tr>
</tbody>
</table>

Note that the planner did not have to replan in order to switch the plans when the state of the environment changed. The changes in expected utilities as a result of the change in the environment caused an implicit switch in plans. This example also illustrates a situation where an error does not have to occur before plans are changed.

### C. Considering Action Costs

So far we have assumed equal costs for all actions. Next, we introduce a scenario, where we add another action, fill-plate which requires the gripper to be holding a utensil as a precondition. There are three plans available when plate-full becomes false and the fork is being used for feeding:

1) fill up the plate, then use the fork again;
2) release the fork and use the spoon; and
3) continue to use the fork.

The selection of the appropriate plan depends not only on the relative probabilities of success but also on the cost of the actions. Consider the case where the cost of performing all the actions including fill-plate are the same. Fig. 4(a) shows the planning behavior when plate-full is initially true but subsequently false. Instead of grasping the spoon [as in Fig. 3(a)], the planner selects fill-plate and then goes back to using the fork. When the cost of fill-plate is made larger than the utility it receives, the planner chooses to select the spoon rather
than filling up the plate [see Fig. 4(b)]. For both these examples, the state of small-spoon is false. If this is turned true and the cost of fill-plate is high, the planner will continue using the fork even when plate-full becomes false.

D. Action Selection under Varying Utility

Decision-theoretic planning implies the ability to weigh the expected utility of an action against its cost and decide whether the action should be taken. In this section, we consider an example of dynamic decision-theoretic action selection, which extends the “static” decision making problem described in the last section. This ability becomes necessary when the expected utility of an action changes either expectedly or unexpectedly.

Consider the following addition to our previous example. The success of picking up food from the plate depends on a condition location-valid. This indicates how accurately the location of the plate is known. When the action locate is performed, location-valid becomes true with a probability of 1. However, with the passing of time, the probability of this proposition being true decreases. For our purposes, this decrease in probability has been simplistically set to be 0.01 at every time step. This raises the necessity for performing the action locate periodically. The question is how often? This can be answered in the decision theoretic framework: The utility for the action locate keeps increasing as the probability of location-valid being true decreases. When this becomes the highest among all the actions, it is performed.4

Fig. 5(a) shows an example of this situation. The expected utility of the locate action gradually rises (as that of use-fork falls) until it has the highest value. Performing the action raises the probability of location-valid being true back to 1, also raising the expected utility of use-fork.

So far, the cost of the sensing action locate has not been considered (it was set to zero in the situation described above). As the cost of the action increases, it is performed less often, as shown in Fig. 5(b) where the cost of locate is 15 and the utility gained from feeding is 20. Now locate is done only when the probability location-valid being true decreases down to zero.

This example highlights two features of this mechanism. First, the ability to accommodate probabilistic information about a proposition. Second, it reacts to expected or unexpected variations in this probability at run-time. This is achieved because the mechanism employs a decision-theoretic process at the level of run-time action selection. The next section shows other examples of reaction to the unexpected.

E. Handling Unexpected Situations

The ability of the planner to react to unexpected situations have been addressed to some extent in the previous sections. Here, we focus on this ability, especially in situations where 1) an action fails to have its expected effect; 2) an unexpected change in the environment causes a switch in plans; and 3) goal utilities are suddenly changed by the user.

In the extreme case, if some other agent moves the plate, making location-valid false with a probability of 1, locate will have to be performed immediately.
Fig. 6(b) shows the reaction to another unexpected change in the environment, this time for the better. Consider the situation first described in Section I where only the spoon is initially on the table. The planner has no alternative but to pick up the spoon. At this point, a fork is unexpectedly located. As described above, the probability of success with the fork is higher when the plate is full. This causes the planner to switch plans, releasing the spoon and using the fork instead.

Another source for an unexpected situation is changes in goals or their utilities. Here again, the planner reacts by changing the action selected. In the example shown in Fig. 7(a), the original goal was to feed the user. During this process, the user asserted another goal of getting a drink with a higher priority (40, as compared to 20 for feeding). The planner reacts to this change by performing the actions necessary to satisfy the higher priority goal before going back to satisfying the lower priority goal of feeding. Again, the entire process was performed simply by picking the action with the highest expected utility.

In Fig. 7(b), the utility of the new goal is the same as the utility of the existing goal. This is not sufficient to cause a change in plans. The goal of bringing a drink has to wait until the goal of feeding the user has been achieved.

F. Resolving Goal Interactions

Goal interactions occur when an action which is desirable in the context of one goal is undesirable in the context of another. Historically, this was first exemplified by the Sussman “anomaly” in the blocks world domain as shown in Fig. 8(a). This simple task could not be optimally planned by STRIPS because of the conflicting goals. The complexity of this problem arises from two facts. First, even though \( (A \text{ on } B) \) is a goal, it should not be the effect of the first action because this will conflict with the subsequent achievement of the goal \( (B \text{ on } C) \). Instead, block A should be cleared from block C first. Next, both the goals, \( A \text{ on } B \) and \( B \text{ on } C \) can be performed. However, the latter must be done before the former. Fig. 8(b) shows how the action selection network can perform the task optimally.

Problems like the Sussman anomaly inspired the design of nonlinear planners [33] which handle goal interactions by explicitly looking for actions that clobber other sub-goals, and then re-ordering 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.

The assignment of goal utilities allows the user to force priorities on goal achievement without regard to consequent clobbering that will require the planner to take additional actions. This illustrates 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.

G. Where Will It Fail?

So far, the results of successfully applying the planner to various situations have been described. This section discusses some domains and behaviors for which the action selection mechanism is inappropriate.

**Where actions do not have immediate effects.** The action selection mechanism assumes that the effects of an action can be...
sensed right away. While true in robotic domains where physical objects are being manipulated, this assumption restricts the domains where the mechanism may be applied.

**Where actions have incremental effects.** Incremental effects are hard to represent as a Boolean proposition. Consider the act of hitting a nail with a hammer. The desired effect is to level the head of the nail against the adjacent surface. This may require repeated executions of the hammering action. This does not mean that all but the final action were failures.

**Where goals are conjunctive.** That is, achieving one without another is like not achieving anything at all. The assignment of utilities to propositions conveys the desirability of achieving all of them, but does not represent the negative desirability of achieving one without achieving another.

**Where sensing is expensive.** The action selection mechanism assumes the existence of sensors that can quickly report the state of the environment. More often than not, sensing, especially image-based sensing, is a compute-intensive process. Action selection should include actions that sense. An example of this was given earlier in this section, where locate is an action that does not have a physical effect on the environment, but nevertheless improves the probability of success of other actions.

**V. CONCLUSIONS**

This paper develops a framework for generating robust planning behavior in uncertain environments. Action selection in this framework is geared toward actions that have been more reliable in generating the desired effects in the past. This requires an explicit representation of uncertainty in the domain knowledge used for planning. The action selection behavior also includes a reactive component that responds to unexpected changes in the state of the environment as well as changes in the desired goals. This is performed through a bidirectional spreading activation mechanism that receives input from the current state of the environment and the goals.

The primary contributions of the work can be summarized as follows:

1) Representation of uncertainty as a probabilistically linked network of propositions and actions.
2) Decision-theoretic action selection based on actions’ expected utilities and execution costs.
3) Spreading activation based planning that can react to unexpected changes in the state of the environment and goal utilities.
4) Coping with directed and undirected loops in the network during forward and backward propagations.

The capabilities of the planner have been illustrated with examples drawn from a robotic domain where the unstructured nature of the environment leads to uncertainty in sensing and action. The presence of a human in the environment makes it necessary to react to changing goals.

The spreading activation architecture is now being incorporated into the Intelligent Machine Architecture [34], a generic modular and distributed software architecture being developed for the control of service and manufacturing robots.

**REFERENCES**


Sugato Bagchi received the Ph.D. degree in electrical engineering from Vanderbilt University, Nashville, TN, in 1995.

He is currently a Research Staff Member at the IBM T. J. Watson Research Center, Yorktown Heights, NY. His current research interest is in knowledge representation frameworks and methodologies for business strategy formulation and analysis. He has contributed subject matter expertise to numerous strategy consulting engagements. His current area of focus is e-marketplace strategies. He is also involved in a study of firms that are using e-business capabilities to define or implement their strategies. In the recent past, he has worked on the design and implementation of a business process modeling and simulation tool, as well as a specialization of this tool for multi-enterprise supply chains.

Gautam Biswas (S’78–M’82–SM’91) received the Ph.D. degree in computer science from Michigan State University, East Lansing, MI, in 1983.

He is currently an Associate Professor of Computer Science and Engineering, and Management of Technology at Vanderbilt University, Nashville, TN. He conducts research in artificial intelligence with primary interests in modeling and analysis of complex systems and their applications to diagnosis and fault-adaptive control. He is also involved in developing simulation based environments for learning and instruction, and developing planning and scheduling techniques for intelligent manufacturing systems. His research is currently supported by DARPA, NSF, ONR, Agilent Labs, and Xerox PARC.

Dr. Biswas has served on the Program Committee of a number of conferences. He was chair of the 1997 IJCAI Workshop on Engineering Problems for Qualitative Reasoning, co-chair of the 1996 Principles of Diagnosis Workshop, the 1999 AAAI Spring Symposium on Hybrid Systems and AI, the 2001 Workshop on Qualitative Reasoning, Senior Program committee for AAAI-97 and AAAI-98, and Technical Committee co-chair for the 2000 IEEE SMC conference. He is a Senior Member of ACM, AAAI, and the Sigma Xi Research Society.

Kazuhiko Kawamura received the Ph.D. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, in 1971.

He is currently a Professor of Electrical Engineering and Computer Engineering and Professor of Management of Technology at Vanderbilt University, Nashville, TN. He is also the Director of Vanderbilt’s Center for Intelligent Systems and Intelligent Robotics Laboratory.

Dr. Kawamura serves as Member of the SMC Society Administrative Committee and General Chair for the 2000 IEEE International Conference on Systems, Man, and Cybernetics in Nashville, TN, October 8–11, 2000.