SRI International

A PROBABILISTIC MODEL FOR UNCERTAIN PROBLEM SOLVING

Technical Note 256

December 1981

By: Arthur M. Farley*
Computer Scientist

Artificial Intelligence Center
Computer Science and Technology Division

The research reported herein was partially supported by ONR Contract No. N00014-81-C-0115.

---

* This research was performed while the author was on leave from the Department of Computer and Information Science, University of Oregon, Eugene, Oregon 97403.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

ABSTRACT

With growing interest in the application of research to problems that arise in real-world contexts, issues raised by consideration of uncertain states and unreliable operators are receiving increased attention in artificial intelligence research. In this paper, a model is presented for dealing with such concerns. The model is a probabilistic generalization of the familiar notion of problem space. The specification of uncertain states and unreliable operators is discussed. Problem-solving search methods are described. The need for information gathering is established. Search methods are generalized to produce tree-structured plans incorporating the use of such operators. Several application domains for our model are discussed.

CONTENTS

ILLUSTRATIONS

I INTRODUCTION

Until recently, most artificial intelligence research on problem solving ignored issues of state uncertainty and operator unreliability. With a growing desire to apply research results in real-world contexts, these topics have begun to receive increased attention. Real-world contexts present an inherent uncertainty stemming from several factors, such as inadequate interpretation of environmental information, unreliable execution of plan actions, and the unforeseen interaction of multiple agents. The spreading use of expert-system technology based on inexact, probabilistic reasoning is one indication of the current interest in and future potential for real-world applications of artificial intelligence.

This paper offers a theoretical framework for addressing issues of problem solving under conditions of uncertain states and unreliable operators. The framework is a straightforward generalization of the familiar state space model [1]. A forward-directed search algorithm analogous to A* [17] is presented. The specification of unreliable operators and the description of uncertain states are discussed. The need for information-gathering operators to control state disunity and provide pragmatic focusing is established; a representation for such operators is proposed. A backward-directed search component is introduced to assist in the planning of information-gathering operators. Aspects of our model of uncertain problem solving are compared with Markov processes and utility-based techniques of decision analysis. The paper concludes with a summary of results, suggestions regarding applications, and a brief discussion of the outstanding issues in real-world problem solving.

II PROBLEM SOLVING

A problem exists for an agent within a task environment when a current situation does not satisfy all aspects of a desired situation. The agent represents situations in the task environment by states. Problem solving refers to an agent's activity in determining a plan (e.g., a sequence of operators) that transforms one (current) state into another (goal) state. The agent can then perform actions in the task environment corresponding to operators of the plan to realize the desired situation. The search for a solution is carried out within a problem space [16]. A problem space (PS) consists of the following components:

State Space: SS -- a set of possible states

Operators: O -- a set of operators, o: SS --> SS

Current State: cs -- a distinguished state of SS

Goal Space: GS -- a distinguished subset of SS.

The current state represents the concurrent situation in the task environment. The solution to a given problem is a sequence of operators o1 ... oK such that oK(...o1(cs)...) = gs, where gs is an element of GS.

States within a PS are modeled and manipulated by an agent in the form of state descriptions. Two representational formalisms are commonly used to describe states of SS. The two are formally equivalent, but are usually manipulated differently; instances of both appear in examples discussed below. One describes a state by a set of propositions (n-ary predicates) indicating properties and relations that are true of the state. When this form of state description is used, operators are often represented as sets of precondition, add and delete propositions. The implicit meaning of this representation is that, if

the set of preconditions is satisfied by a given state description, then the operator may be applied; a new state description is produced by removing propositions of the delete set from the given state and forming the union of the remainder with the add set. The second form of state description that is frequently used is a set of attribute-value pairs. For a state description, a mentioned attribute has the specified value in the state(s) being represented. This form of state description can be reduced to a vector of values with an implicit attribute name for each vector position (as in a relational data base). Operator preconditions are represented as constraints on attribute values; effects are represented as functions expressing new attribute values in terms of given values.

By specifying ranges for predicate arguments or attribute values, we determine a (superset of the) state space. The state space may be further constrained by state axioms. State axioms describe allowable or necessary relationships among value aspects of a state description. For example, two attributes may have integer value ranges; a state axiom may specify that in every allowable state one of the values must be greater than twice the other. Another state axiom may indicate that a new predicate or attribute value pair can be inferred from others currently in a state. For example, a blocks-world state axiom may assert that, if block A is on B and B is above block C, then the proposition that A is above C can be added to the state.

Given either form of state and operator description in a PS, there exists a straightforward model of problem solving viewed as a process of state space search. The process starts at the current state and applies possible operators to resultant states until reaching a state in GS. The following algorithm specifies this forward-directed search method:

Algorithm SOLVE(SS,O,cs,GS):

(1) Set FRINGE to { cs }.
(2) Select a state s in FRINGE.
(3) If [s in GS] then Retrieve Solution and Stop.
(4) Otherwise, Remove s from FRINGE and put it in OLD.
(5) Apply all possible operators in O to s producing NEW.

(6) Add each state in NEW that is not in FRINGE or OLD to FRINGE.

(7) Return to Step 2.

As an operator o is applied to state s, we associate with each new state ns two values: a predecessor pred(ns), being equal to s, and a preceding operator preop(ns), being equal to o. These allow straightforward retrieval of a solution in Step 3. Several instances of a given state may be produced (by differing operator sequences) during problem-solving search; with each maintained instance, one (e.g., first or least-cost) predecessor is associated [17]. States in NEW that are incorporated into the search tree are associated with state s as successors(s).

The nondeterministic selection of a next state in Step 2 provides opportunity for the use of heuristics to guide the otherwise blind, forward-directed search. One means of introducing heuristics is to associate with each state in FRINGE the result of an evaluation function applied to that state. A common form of evaluation function is $f(s) = c(s) + h(s)$ for s a state in FRINGE, where f maps s to a numeric (integer or real) value [17]. The value of c(s) represents the best (lowest) cost of a known sequence of operators reaching s from cs. The value of h(s) represents an estimate of the cost to reach an element of GS from s. Thus, f(s) represents an estimate of the cost of a solution to the given problem, with the solution constrained to go through state s. It is natural to select a next state s from FRINGE, which minimizes this estimate. It has been proved that if this is done and h(s) is a lower bound on the actual cost of reaching a state in GS from s, then Algorithm SOLVE is guaranteed to find a minimum-cost solution to a given problem if a solution is indeed possible [17]. Algorithm SOLVE under these conditions is called Algorithm A*. The valued property of A*, that of always finding a minimum cost solution when a solution exists, is called admissibility.

III INTRODUCING UNCERTAINTY

The preceding model of problem solving is severely limited in its applicability. Most importantly, it is not a suitable representation for solving problems that occur in real-world task environments. Several properties of the latter, enumerated below, make this conclusion apparent:

* The environment is ongoing (prior to goal state specification)
* A state only partially represents any environmental situation
* There is likely error in transduction of situational information
* Operator specifications are incomplete (i.e., have side effects)
* Actions corresponding to operators are executed unreliably
* Other systems may affect the task environment.

All the above properties of real-world task environments indicate that problem solving must deal with issues of imprecision and uncertainty if it is to produce effective solutions for such contexts. In the following discussion we shall propose extensions of the earlier problem-solving model that take into account the existence of uncertain states and unreliable operators.

Problem solving under conditions of uncertainty takes place within an uncertain problem space. An uncertain problem space (UPS) is comprised of the following aspects:

State Space: SS -- a set of possible states

Possible States Space: PSS -- the set of sets of states in SS (i.e., 2^SS)

Uncertain State Space: USS -- as PSS, associating probabilities summing to 1.0 with component states of an element of PSS

Unreliable Operators: UO -- a set of operators, uo: USS --> USS

Uncertain Current State: ucs -- a distinguished element of USS

Goal Space: GS -- a distinguished element of PSS

Solution Threshold: ST -- a number, 0 =< ST =< 1.

A problem specified in an uncertain problem space is an uncertain problem. Assessment of the present situation in the task environment is represented by ucs. As that environment is ongoing, a problem solver is typically uncertain about those aspects of the present situation that are relevant to achieving a new goal. GS is defined, as in the certain case, to be a subset of the states in SS.

We initially define a solution to an uncertain problem to be a sequence of unreliable operators uo1 ... uoK with uoK(...uo1(ucs)...)=ugs, such that the sum of the probabilities of states in ugs that are also in GS is greater than or equal to ST. With each uncertain state us, we associate a degree of goal space satisfaction dsat(us) equal to the sum of the probabilities of component states of us that are in GS (i.e., that satisfy goal space criteria). Solving an uncertain problem consists of determining a sequence of unreliable operators that produce an uncertain state us, such that dsat(us) >= ST. The notion is that an uncertain problem is solved by a sequence of unreliable operators that transforms an uncertain current state into one of a set of goal states with some minimum (threshold) probability.

An uncertain problem space represents an uncertain situation as a set of possible states having probability prob(s) associated with each state s of that set. Thus, an uncertain state corresponds to a set of representable "possible worlds," with the likelihood that each of those worlds is the actual world. An unreliable operator maps one uncertain state to another. Note that an uncertain problem reduces to a certain problem when ucs consists of one state of SS with probability 1.0, all operators map a single state of SS to another state of SS with

probability 1.0 (i.e., they are reliable), and ST is equal to 1.0. Throughout our discussions, we use probability in the sense of "best estimate of likelihood" and do not imply knowledge by the problem solver as to actual probabilities. We do assume that probability estimates are assigned consistently, i.e., sum to 1.0 for each uncertain state.

In a (certain) PS, an operator is applied only to states satisfying its preconditions; for such states the operator produces its (desired) effects. In a UPS, an unreliable operator must be applicable to uncertain states containing some component states that satisfy and some that do not satisfy its (desired) preconditions. An unreliable operator must be defined as to its effects given any state in SS. This can be done by specifying operators as sets of precondition-effects pairs such that the precondition aspects partition the states of SS. For each specified precondition aspect, a set of effects with associated probabilities is defined. Figure 1 describes this framework for specifying unreliable operators. Given uncertain state us, the uncertain state resulting from the application of an unreliable operator can be computed by combining the effects for each component state in us; resultant component state probabilities reflect the likelihood of effects and prior components in us. Figure 2 presents function APPLY, which determines the uncertain state resulting from application of an unreliable operator to an uncertain state.

As specified, each operator description is equivalent to a finite-state Markov process [12,13], expressed as mappings between state descriptions rather than as transitions between states (in a matrix). Application of a sequence of such operators represents a discrete-time, controllable Markov process. The process is controllable in that the probabilistic state transition is determined by the selected operator. Function APPLY describes a procedure for computing the next state configuration of a controllable, unobserved Markov process. The process is unobserved in that we assume no means for determining which of the possible states actually exists before or after application of an unreliable operator.

UO = [(pre, [(eff, peff)])],

where [...] indicates a nonempty set of the enclosed elements,

such that

(i) pre and eff are state descriptions;

(ii) peff is an effect probability, 0 =< peff =< 1.0;

(iii) the set of preconditions pre partitions SS;

(iv) the sum of peff for given pre equals 1.0.

Associated are the following access functions:

pre(UO,i) :: the ith precondition of UO;

eff(UO,i,j) :: the jth effect associated with the
ith precondition of UO;

peff(UO,i,j) :: the probability of that jth effect.

Figure 1 Specification Schema for an Unreliable Operator UO

```
function APPLY(US,UO)
  begin
    Set rus to the empty set;
    For each s in US
      begin
        Determine i such that s satisfies pre(UO,i);
        For each j such that eff(UO,i,j) exists
          begin
            Generate state ns, evaluating eff(UO,i,j) in terms of s;
            Set prob(ns) to prob(s) * peff(UO,i,j);
            If [ns = os in rus] then increment prob(os) by prob(ns)
              else place ns in rus
          end
      end;
    Return(rus)
  end
```

Figure 2 Function APPLY, Which Applies Unreliable Operator UO to Uncertain State US and Returns the Resultant Uncertain State (rus)

Most optimization results in Markov control and decision analysis rest upon the assumption of complete observability (as in [2]). These results take the form of policies that specify for each individual state which operator is to be applied to maximize expected utility over either an infinite or finite number of subsequent transitions. These results assume a utility function defined over the possible states. Approximate solutions in the infinite case are expressed in terms of an iterative (or recursive), convergent, policy adjustment process [5,12]. In the finite case, an optimal policy for n state transitions is expressed recursively in terms of optimal policies for n-1 transitions from all states directly reachable from the initial state [12]. Such a result is easily expressed, but implies a complete search of all possible futures to a given depth. Various techniques of dynamic programming, such as branch and bound [24], have been proposed to reduce the exponential, computational complexity. Similar results have been expressed for partially observable processes [22]; the attempt is made to reduce the complexity by partitioning the space of uncertain states into regions of equal expected utility. These results appear to be extendable to the totally unobserved case as well.

We are concerned here with a slightly different problem. Expressed in the terminology of above results, we want to determine the least cost of a sequence of transitions resulting in an uncertain (unobserved) state that satisfies goal space criteria with probability greater than a given solution threshold. Prior research has discussed least-cost operator selection policies for completely observed Mankov processes [4]. A recent result addresses a related issue in a completely observable context, that of minimizing the expected number transitions to reach a certain state from a given initial state [11]. Controls in that case consisted of filters limiting the possible outcome states of a transition.

We would like to determine heuristic methods that are applicable in a UPS. Our model of the latter allows for the definition of USOLVE as a straightforward generalization of the state space search method

presented as Algorithm SOLVE. The halting condition in Step 3 becomes one of verifying satisfaction of the threshold condition required by ST (i.e., dsat(us) >= ST). Values for successors, pred, and preop can be associated with uncertain states during search as in the certain case, facilitating retrieval of the solution plan. Furthermore, there is an analogous definition of a state evaluation function guaranteeing admissibility of USOLVE. Given the definition of a suitable h function in a PS, we can specify an evaluation function for uncertain states that guarantees admissibility of USOLVE in the corresponding uncertain problem space. For an uncertain state us in FRINGE, we define the evaluation function uf(us) = uc(us) + uh(us). The value of uc(us) is the sum of expected costs of applying operators on the path from ucs to us, as in the certain case. Each unreliable operator definition must associate costs with each possible transition. The expected cost of applying an operator to an arbitrary uncertain state can then be determined in a straightforward manner. We could define uh(us) to be min[h(s)] over component states of us; however, if dsat(us) > 0 then uh(us) would equal 0. A better definition of uh would be the minimum of max[h(s)], maximized over states s from subsets of component states of us having combined probability greater than or equal to ST, where the minimum is taken over all such sets. By either definition, uh(us) would be less than the actual cost to reach an uncertain solution state. As such, USOLVE would be admissible.

IV REPRESENTING UNCERTAIN PROBLEM SPACES

The definition of uncertain problem spaces given above allows straightforward generalization of our (certain) notions of problem, solution, and problem solving. However, several representational issues remain to be considered. In this section we discuss the representation of uncertain states, using several examples to illustrate important issues.

One interesting class of uncertain problems consists of those often studied by probability theorists. Although reflecting uncertain aspects not of real-world contexts but of combinatorial spaces, these problems are straightforwardly modeled by uncertain problem spaces. Figure 3 presents an example of an uncertain problem space modeling the task environment for a class of problems concerned with the random drawing of balls from an urn that contains balls of two colors. The extracted balls are placed in a box. A state represents the number of balls of each color in both the urn and the box, as well as the color of the last ball drawn. The action of randomly drawing a ball is directly representable as an unreliable operator. The specification of DRAW illustrates that the probabilities of effects associated with a given precondition, can be variable, depending upon aspects of the state to which the operator is being applied. All that is required is local consistency in that the effect probabilities for any component state must sum to 1.0. The example shows DRAW being applied to an uncertain current state in which no balls have yet been drawn. It is not known whether the urn contains 70 balls of one color and 30 of another or an equal number of each color; however, the former is considered to be more probable. If the goal state is the presence of one or more balls of color1 in the box with ST equal to 0.60, then one application of DRAW solves the problem; dsat(UCS′) is equal to 0.62.

An element s of SS is a vector of the form (U1,U2,B1,B2,CL),

where U1 and U2 are the number of balls in the urn of color1 and color2, respectively;

B1 and B2 are the same for the box;

CL is the color of the last ball drawn (0, if none).

PSS and USS are defined implicitly.

Figure 3(a) State Space SS

```
DRAW = { ( (U1 + U2 > 0) , {((U1-1,U2,B1+1,B2,1), U1/(U1+U2)),
                            ((U1,U2-1,B1,B2+1,2), U2/(U1+U2))})

         ( (U1 + U2 = 0) , {((0,0,B1,B2,0), 1.0)})  }
```

Figure 3(b) Specification of the Uncertain Operator DRAW

```
UCS = { ((70,30,0,0,0), 0.6), ((50,50,0,0,0), 0.4) } .

DRAW(UCS) = { ((69,30,1,0,1), 0.42), ((70,29,0,1,2), 0.18),
              ((49,50,1,0,1), 0.20), ((50,49,0,1,2), 0.20)  } = UCS'
```

Figure 3(c) Application of DRAW to Uncertain Current State UCS

Figure 3 Specification of a UPS for Random Ball Drawings

The primary motivation for considering uncertain problem spaces in artificial intelligence derives from a desire to deal effectively with real-world contexts. Figure 4 presents an example of an uncertain problem space modeling a simple robotics task environment. The robot operates in a world in which a block (or assembly part) can be moved about on a tabletop. A position on the tabletop is represented by a pair (x,y) of integer coordinates. A state in the problem space consists of a single proposition representing that (a distinguished point of) the block is at (i.e., within an epsilon of) or near (i.e., within one prescribed unit of) a particular position. The parameterized movement operator MOVE(x1,y1,x2,y2) is unreliable. If the block is known to be at (x1,y1), the operator may drop the block as it picks it up or may only place it near (x2,y2). If the block is known only to be near (x1,y1), the operator may miss it altogether; this will definitely happen when the block is not at or near (x1,y1). Figure 4(c) demonstrates that the better our knowledge is as to the position of the block, the more effective is the operator during problem-solving search. If the goal is for the block to be at or near position (50,50) (i.e., GS = { (AT B, 50, 50), (NEAR B, 50, 50) }), then dsat(URS) equals .815 while dsat(URS´) only equals .635.

As noted earlier, state spaces are modeled and manipulated as state descriptions. A given state description may represent a set (i.e., more than one) of states from SS. This implicit state uncertainty is inherent in many state descriptions; actually, it has been common in earlier certain-case approaches to problem solving. In our block example, the predicate NEAR introduces one form of implicit state uncertainty. Being equivalent to a disjunction of other, more specific propositions, an instance of NEAR represents a set of possible states. Such uncertainty can occur in attribute value state representations if a general value is defined for a given attribute that denotes several values in the range of that attribute. Implicit state uncertainty may also be introduced by incomplete state specifications. In the attribute value formalization, only a subset of attributes may be given values; an unspecified (i.e., "don´t care") attribute may take on any value in its

An element of SS is of the form

```
   (AT B, X, Y) or (NEAR B, X, Y),    for 0 =< X,Y =< 100,
where
              (NEAR B, X, Y) == (AT B, X, Y-1) or
                                (AT B, X, Y+1) or
                                (AT B, X-1, Y) or
                                (AT B, X+1, Y)  .
```

PSS and USS are defined implicitly.

Figure 4(a) State Space SS

```
MOVE(x1,y1,x2,y2)  ==  { ( (AT B, x1, y1) , { ((AT B, x2, y2), 0.85)
                                              ((NEAR B, x2, y2), 0.10)
                                              ((NEAR B, x1, y1), 0.05)})
                         ( (NEAR B, x1, y1) , { ((AT B, x2, y2), 0.45)
                                                ((NEAR B, x2, y2), 0.05)
                                                ((NEAR B, x1, y1), .5)})
                         (OTHERWISE, NO CHANGE) }
```

Figure 4(b) Specification of Unreliable Operator MOVE

```
UCS = { (AT B, 20, 20), 0.7), (NEAR B, 20, 20), 0.3) }

          MOVE(20,20,50,50)
UCS  ------------------------->{((AT B, 50, 50), 0.73)
                                ((NEAR B, 50, 50), 0.085),
                                ((NEAR B, 20, 20), 0.185) } = URS

UCS' = { ((AT B, 20, 20), 0.3), ((NEAR B, 20, 20), 0.7) }

          MOVE(20, 20, 50, 50)
UCS' ------------------------->{((AT B, 50, 50), 0.57)
                                ((NEAR B, 50, 50), 0.065)
                                ((NEAR B, 20, 20), 0.365)  } = URS'
```

Figure 4(c) Application of MOVE to a Pair of Uncertain States

Figure 4 Specification of a UPS for a Simple, Unreliable Block World

range consistent with state axioms associated with the state space. Similarly, in a proposition-based state space, certain properties or relations may not be specified. When state descriptions are allowed to indicate aspects that are not true of the states they represent, large uncertain states can be specified by relatively few disallowed attribute values or negated propositions.

Implicit state uncertainty provides economy in specifying an uncertain problem space to USOLVE and in carrying out problem solving search by USOLVE. However, it does raise several difficulties with regard to intersecting state descriptions. Within a given uncertain state, it is desirable to have disjoint component state descriptions; in fact, all definitions and algorithms discussed thus far in our paper assume this property of uncertain states. For example, if the probabilities of component state descriptions are to sum to 1.0, the components must be disjoint. Even if we require that ucs be specified as a set of disjoint state descriptions, application of an unreliable operator still may require USOLVE to consider subsets of the implicit set of states associated with a given component state description when matching preconditions or computing effects. One component may satisfy several precondition partitions; application of an operator may cause intersecting components to be produced in the resultant state. In these cases, USOLVE must expand the component state to establish a set of state descriptions at the required level of representational detail and assign probabilities to the resultant, more specific components. State axioms will be used to select allowable component states. Probabilities are assigned according to the implicit relative cardinalities of the new components, the probabilities of original components, and any relative-probability axioms that have been specified. Relative-probability axioms indicate a priori relative likelihoods among value aspects of allowable sets of states. If no relative-probability axioms have been specified for a given aspect, a uniform distribution is assumed.

Figure 5 illustrates these notions, applying new instances of MOVE in the UPS described in Figure 4. Consider applying MOVE(21,20,50,50)

to the uncertain state UCS of Figure 4(c). Attempts to match preconditions of the operator cause the state (NEAR B, 20, 20) to be expanded. Assuming no relative-probability axioms have been specified, a uniform distribution of implicit states results in the precondition (AT B, 21, 20) being matched by a component state with probability 0.075. Precondition (NEAR B, 21, 20) is matched by components with probability equal to 0.7, being equivalent in this case to (AT B, 20, 20). The other implicit states represented by the instance of NEAR are states not affected by MOVE. The resultant uncertain state is shown in Figure 5(a). Two state descriptions intersect if their implicit state sets have states in common. When the effects of an unreliable operator are evaluated, intersecting component states may be produced. Figure 5 presents an example of this phenomenon, showing MOVE(20,20,21,20), as applied to UCS of Figure 4(c). The resultant uncertain state includes two intersecting components: (AT B, 21, 20) and (NEAR B, 20, 20). The implicit states represented by NEAR must be explicitly represented. If we again assume uniform distribution of probabilities, Figure 5(b) depicts the uncertain state produced by this instance of MOVE.

The use of state descriptions makes it possible to encourage in the specification of uncertain problem spaces and the representation of uncertain states during problem solving. However, the concomitant implicit state uncertainty does necessitate operations that can detect and isolate intersecting components and reestablish a disjoint uncertain state at a level of specificity required for operator application. In function APPLY presented as Figure 2, an effect expansion procedure must be activated just prior to returning uncertain-state rus. Precondition expansion may be required when an attempt is made to ascertain the precondition satisfied by a given component state description; a component must be expanded so that each resultant component satisfies one and only one precondition.

```
UCS == { ((AT B, 20, 20), 0.7), ((AT B, 19, 20), 0.075),
         ((AT B, 21, 20), 0.075), ((AT B, 20, 19), 0.075)
         ((AT B, 20, 21), 0.075) }
```

```
                MOVE(21,20,50,50)
UCS ──────────────────────────────► { ((AT B, 50, 50), 0.379)
                                      ((NEAR B, 50, 50), 0.043)
                                      ((NEAR B, 21, 20), 0.353)
                                      ((AT B, 19, 20), 0.075)
                                      ((AT B, 20, 21), 0.075)
                                      ((AT B, 20, 19), 0.075) }
```

Figure 5(a) Dealing with Implicit Uncertainty in State Descriptions

```
                MOVE(20,20,21,20)
UCS ──────────────────────────────► { ((AT B, 21, 20), 0.73)
                                      ((NEAR B, 21, 20), 0.085)
                                      ((NEAR B, 20, 20), 0.185) } = US
```

```
US == { ((AT B, 21, 20), 0.777), ((NEAR B, 21, 20), 0.085)
        ((AT B, 19, 20), 0.046), ((AT B, 20, 21), 0.046)
        ((AT B, 20, 19), 0,046)  }
```

Figure 5(b) Applying MOVE to UCS of Figure 4 with Effect Expansion

Figure 5 Dealing with Implicit Uncertainty in State Descriptions

V STATE DISUNITY AND PRAGMATIC FOCUSING

Now that the representation of uncertain problem spaces has been discussed, attention can be directed to issues of problem solving in such problem spaces. In this section we respond to a critical question concerning uncertain problem solving: can uncertain problems having high solution thresholds be solved? This obviously depends on ucs and the reliability properties of available operators. One must realize that uncertainty is compounded by application of unreliable operators. In the model thus far proposed, uncertain states are mapped to other uncertain states by unreliable operators that could normally be expected to introduce even greater uncertainty. Execution of USOLVE in the UPS of Figure 3 can determine a minimum-draw solution for any problem requiring k or more balls of either color to be in the box with any desired ST (for any k less than or equal to 30). Formulation of a plan guaranteeing exactly k balls of either color to be in the box is another story. As DRAW is applied repeatedly, the unreliability of the operator leads to greater disunity (increasing entropy) in resultant uncertain states. The level of disunity reflects the proliferation of component states in an uncertain state and the failure of any one component to acquire a high probability. In our example, it will never be more likely that exactly one ball of colorl is in the box than it is after the first DRAW; after two DRAWs the probability is reduced to near 0.40. In general, it may remain possible to solve problems having low solution thresholds or large goal spaces (e.g., that allow a wide range of values on goal attributes) after a sequence of unreliable operators has been applied. It may be expected to become increasingly difficult, if not impossible, to solve uncertain problems having reasonably constrained goal spaces with high solution thresholds when application of several unreliable operators is required.

The process of solving uncertain problems should be sensitive to the degree of disunity in the uncertain states considered during the search for a solution. As such, the following measure of disunity is proposed. Let the disunity of an uncertain state us, disu(us), be equal to one less than the sum of i*prob(si) for 1=<i<=k, where us is composed of states s1,...,sk such that prob(s1)>=prob(s2)>=...>=prob(sk). This measure of disunity has the following suitable properties: disu(us) is minimized (i.e. equals 0) when us consists of a single state of SS; uncertain states with fewer component states tend to have lower measures of disunity; for uncertain states having the same number of components, the more equal the probabilities of those components, the higher is the measure of disunity; disu(us) is maximized (i.e., equals (k-1)/2) for us having k components when those components are equally probable. This measure of disunity is one less than the expected number of component states considered by a search for a component matching the task environment, given that the components are considered sequentially in order of decreasing probability [14]. When disu(us) equals 0, us is said to be a certain state. The disunity measure should be evaluated in terms of states, not state descriptions, if it is to be totally correct--and possibly at the most specific level of state description currently required, if it is to be most meaningful. The entropy [21] of an uncertain state us, equaling the sum of -i*log(prob(si)) with probabilities of components ordered as above, has properties similar to disu(us) and could be used as a measure of disunity as well.

Algorithm USOLVE could consider the value of disu(us) when selecting a next state. A selected uncertain state with a high level of disunity need not always be developed immediately, only tested for its degree of goal space satisfaction. During search, USOLVE may eventually apply operators to such states. What are needed are operators that can control (i.e., reduce) the level of disunity in uncertain states. USOLVE then could apply such operators to a selected state that has unacceptably high disunity. However, even when disunity is kept at a relatively low level, unfavorable circumstances can arise. Consider the aspects of an abstract, uncertain problem space presented in Figure 6.

Assume that only those aspects of the problem space specified there are relevant to solving the given problem; aspects are specified directly in terms of the names of states occurring in SS. The value of disu(ucs) is only 0.3, indicating that disunity is quite low, approaching the minimum possible. Yet, as shown by Figure 6(b), the goal state S3 can not be satisfied to the degree required by ST. This is true even though operators A and B are each capable of taking a different one of the two component states of UCS to goal state S3 with probability greater than that required by the solution threshold ST. The probabilities associated with components of UCS do not allow either operator to achieve S3 with sufficient likelihood.

Not only must a solution plan control disunity, it must increase the probabilities of component states that can lead to high degrees of satisfaction for states in GS. We assign the term pragmatic focusing to activities that serve to increase the probabilities of component states that satisfy preconditions of operators capable of reliably producing states in a given goal space. In certain problem spaces, pragmatic focusing corresponds to the satisfaction of subgoals that have been generated by backward-directed, means-ends analysis [16]. What is needed is a means of pragmatic focusing in uncertain problem solving and plan execution.

```
SS = {S1,S2,S3,S4,...};

PSS and USS are defined implicitly;

UO = { A = { (S1, {(S3, 0.9), (S4, 0.1)}),
             (S2, {(S3, 0.1), (S4, 0.9)}),
             (S3, {(S4, 1.0)}), (S4, {(S4, 1.0)}),
             (OTHERWISE, {(S4, 1.0)})  }

       B = { (S1, {(S3, 0.1), (S4, 0.9)}),
             (S2, {(S3, 0.9), (S4, 0.1)}),
             (S3, {(S4, 1.0)}), (S4, {(S4, 1.0)}),
             (OTHERWISE, {(S4, 1.0)})   }

                         .

                         .

                         .      };

UCS = { (S1, 0.7}, (S2, 0.3) };

GS  = { S3 };

ST  =  0.8.
```

Figure 6(a) Specification of the Uncertain Problem Space

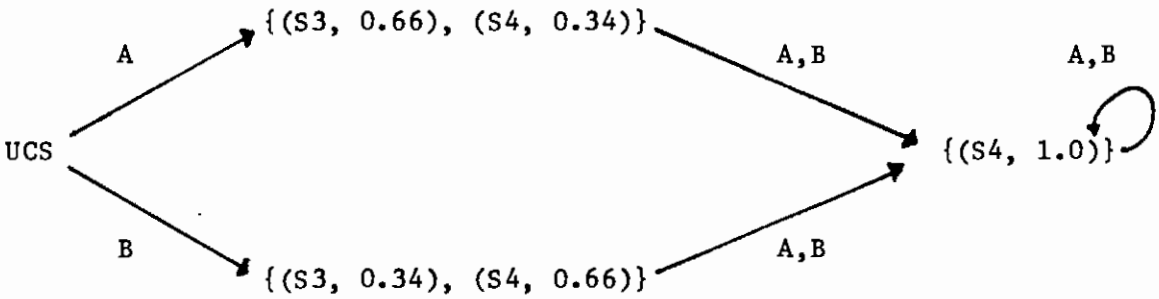

Figure 6(b) Application of Operators A and B from UCS

Figure 6 An Example Demonstrating the Need for Pragmatic Focusing

VI INFORMATION GATHERING

We have established the need for operators that can control state disunity and provide pragmatic focusing in uncertain problem-solving. The class of information-gathering operators is relevant to both capabilities. An information-gathering operator (IGO) is one that can determine state information representative of the prevailing situation in the task environment when it is applied during plan execution. In this section we propose a representation for information-gathering operators. Our initial assumption is that information-gathering operators are perfect, i.e., that the result of an information-gathering operator will be true of the task environment when that operator is applied during plan execution. Furthermore, we assume that the IGOs attain this performance regardless of the current state. Later in this section, we discuss a representation for imperfect information-gathering operators whose performance is affected according to existing conditions. In the next section, we describe search methods that formulate solution plans incorporating the use of information-gathering operators.

In the course of problem solving, planned application of an information-gathering operator maps the one uncertain state to which it is applied into several uncertain states, one for each possible result (answer) of the operator. Thus, a plan solving an uncertain problem that employs information-gathering operators is not a simple list (or sequence) simple sequence of operators, but a tree. Each possible result of an information-gathering operator produces a corresponding uncertain state, eliminating component states that are inconsistent with the result--while adjusting the probabilities of consistent component states according to Baye's Rule [2]. We assume that the possible results of an IGO partition of the state space SS.

With each result of an information-gathering operator we can associate an a priori probability at planning time. That probability is the sum of the probabilities of component states of the uncertain state to which the operator is applied that are consistent with the result. With each uncertain state us, we associate a probability uprob(us) equal to the product of the a priori probabilities of IGO results occurring between us and ucs, or equal to 1.0 if no information-gathering operator has been applied. This probability indicates the likelihood that the uncertain state will be on the path of the solution plan actually followed at execution time.

Figure 7 returns to our example of Figure 3, in which balls are drawn randomly from an urn and placed in a box. The information-gathering operator QCL, which can reliably determine the color of the last ball drawn is introduced. The result of applying QCL to the state UCS′ produced by an initial draw is shown. This produces two uncertain states, UCS′′ and UCS′′′, corresponding to whether the first ball drawn was color1 or color2. The following probabilities are associated with the new uncertain states: uprob(UCS′′) = 0.62 and uprob(UCS′′′) = 0.38. Not only are some states of uncertain state UCS′ eliminated from each of the new uncertain states, but the results of QCL cause modifications in estimated probabilities as to the current urn situation (e.g., when color2 is drawn, estimates that color1 is prevalent decrease in UCS′′′). This occurs as the probabilities of remaining states are normalized to again sum to 1. These new probabilities correspond to the conditional probabilities of each component state, given the associated result of the information-gathering operator. Applying QCL reduces the disunity in the resultant uncertain states; disu(UCS′) = 1.14, while disu(UCS′′) = 0.32 and disu(UCS′′′) = 0.48. If we now apply DRAW to UCS′′′, producing UCS′′′′, the goal of having exactly one ball of color1 in the box is satisfied by our overall plan with a probability of approximately 0.85 (= 0.62*1 + 0.38*(0.34 + 0.26)). With QCL, goal spaces requiring exactly k balls of either color can be satisfied with any ST for any k less than or equal to 30.

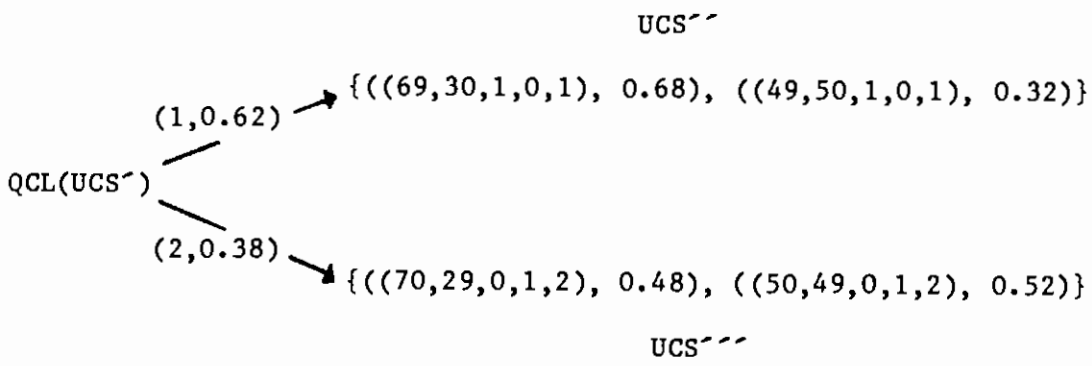

Figure 7(a) Applying QCL to UCS' of Figure 3(c)

$$\text{DRAW}(\text{UCS}''') = \{((69,29,1,1,1),\ 0.34),\ ((70,28,0,2,2),\ 0.14),\ ((49,49,1,1,1),\ 0.26),\ ((50,48,0,2,2),\ 0.26)\} = \text{UCS}''''$$

Figure 7(b) Applying DRAW to UCS'''

Figure 7 Introducing Information-Gathering Operator QCL

Perfect-information gathering operators expand the domain of problems that can be solved with relatively high solution thresholds. Our example above demonstrates this for the case in which there is only one operator. In cases where several (or many) operators are available, the importance of information gathering could be expected to increase. Information gathering segments an uncertain state into subcases for which the most appropriate operators can then be selected. The resultant plans acquire the flavor of planning for contingencies. A solution is a plan that covers enough of the possible contingencies with sufficient reliability to guarantee goal space satisfaction to the extent required by the solution threshold.

Figure 8 demonstrates this idea, applying information-gathering operators in the UPSs defined in Figures 4 and 6. Operator QXY can determine where the block is located on the table within the robotics task environment. Figure 8(a) illustrates application of QXY to uncertain state UCS′ of Figure 4(c). Note our assumption of equal probabilities for the four positions represented by proposition NEAR, as discussed earlier. The parameters of MOVE can now be adjusted for each subcase to realize optimal performance. Repeated application of QXY followed by appropriately specified MOVEs in unsuccessful cases could produce a certainty of final position greater than the 0.85 performance bound of a single, optimally parameterized MOVE. Information-gathering operator QS1 is applied in the UPS defined in Figure 6. It can determine whether the present situation is equivalent to state S1, returning Y if so and otherwise N. With QS1, the goal state S3 can be realized with probability 0.90 by an appropriate selection of operators, as shown in the plan of Figure 8(b). This plan attains the satisfaction limit imposed by the reliabilities of available operators.

We have established several roles that information-gathering operators perform in uncertain problem solving. In addition, we have determined the effects of such operators upon that process. Now let us consider imperfect IGOs (IIGOs) whose performance is dependent upon conditions of the state in which they are applied. Figure 9 presents a framework for specifying such an operator. Once again a set of preconditions is defined that partitions the state space SS; the set of possible results also partitions SS. For each precondition, a confusion table among possible results for that precondition is described as a set of triples (r1,r2,cprob). For a given precondition, each triple indicates by cprob how likely IIGO is to produce result r1 when the current situation actually is consistent with r2. Figure 10 presents two functions to be used in applying an imperfect IGO to a given uncertain state. Precondition and result expansion may be required in both functions to ascertain the implicit aspects of a given component state description that satisfy a given precondition or are consistent with a given result. An imperfect IGO could be applied by first

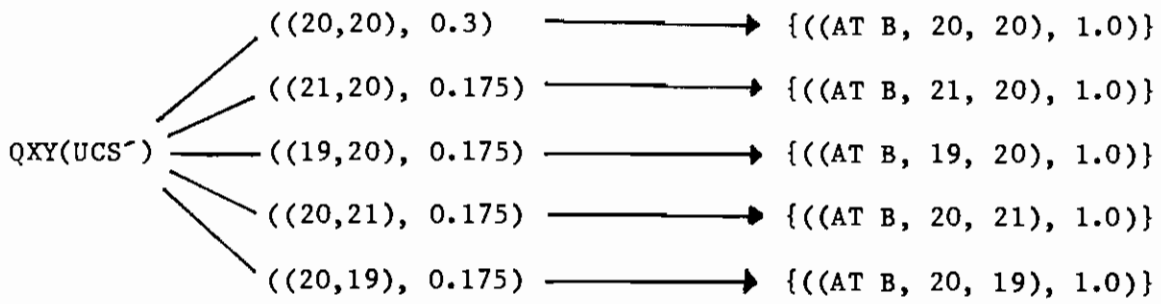

Figure 8(a) Applying QXY to UCS´ in the UPS of Figure 4

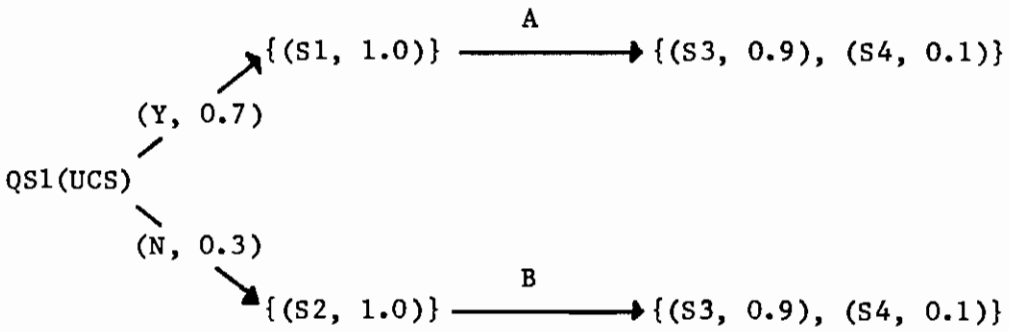

Figure 8(b) Applying QS1 in the UPS of Figure 6

Figure 8 Pragmatic Focusing in the UPSs of Figures 4 and 6

identifying all the results that have positive probabilities, possibly using function RESPROB and applicable pruning knowledge; function RESUS could then be applied for each such result to determine its associated consequent uncertain state.

Figures 11 and 12 demonstrate the definition and application of imperfect IGOs, illustrating certain of their properties. In the first example, an uncertain list-processing problem space is defined. A state of SS is a list of letter elements A, B, or C. An instance UCS of an uncertain state is presented, consisting of four elements of SS. The

IIGO = [(pre, [(r1, r2, cprob)])],

where [...] indicates a nonempty set of the enclosed elements,

such that

(i) pre is a state description;

(ii) the set of preconditions pre partitions SS;

(iii) r1 and r2 are two (possibly the same) results of IIGO;

(iv) cprob is a confusion probability, 0 =< cprob =< 1.0.

Associated are the functions:

pre(IIGO,i) :: the ith precondition of IIGO;

cprob(IIGO,i,r1,r2) :: the probability that IIGO yields result r1 when the situation is actually consistent with r2, given precondition i; if (r1, r2, cprob) is not specified, the corresponding cprob(IIGO,i,r1,r2) is 0.0.

The specified values of cprob are such that, for a given IIGO and its ith precondition,

the sum of cprob(IIGO,i,r,r2) = 1.0,
when summed over all results r,
for each possible result r2.

Figure 9 Specification Schema for an Imperfect Information-Gathering Operator IIGO

imperfect information-gathering operator Qlst is defined, which returns as its result an indication as to the first element of the current list. Its definition reflects the notion that A and B are similar and may be confused; B and C also share common features and may be confused, but less often. Figure 11(c) shows the consequent uncertain states after applying Qlst to UCS. This result can be compared with one that would be produced by the corresponding perfect IGO. Such an operator would return result A with probability 0.45, B with 0.30, and C with 0.25; the uncertain states associated with results B and C would actually be

```
function RESPROB(IIGO, RES, US)
  begin
    Set rprob to 0;
    For each possible result r do
      begin
        For each state s in US consistent with result r do
          begin
            Set pi to i such that s satisfies pre(IIGO,i);
            Set scprob to cprob(IIGO,pi,RES,r);
            Set rprob to rprob + (scprob * prob(s))
          end
      end;
    Return(rprob)
  end
```

Figure 10(a) Functions for Applying Imperfect Information-Gathering Operators

```
function RESUS(IIGO, RES, US)
  begin
    Set rprob to RESPROB(IIGO,RES,US);
    Set nus to { };
    Set uprob(nus) to rprob * uprob(US);
    For each state s in US do
      begin
        Set r to result with which s is consistent;
        Set pi to i such that s satisfies pre(IIGO,i);
        If [cprob(IIGO,pi,RES,r) > 0.0] then
          begin
            Set ns to a copy of state s;
            Set prob(ns) to (prob(s) * cprob(IIGO,pi,RES,r)) / rprob;
            Set nus to nus + { ns }
          end
      end;
      Return(nus)
  end
```

Figure 10(b) A Function Determining the Uncertain State Associated with a Given Response

Figure 10 Functions for Applying Imperfect Information-Gathering Operators

certain. With Qlst, these two uncertain states contain all four components; however, their disunity measures are less than 1.0 because of the relatively low confusability of results.

```
Each state of SS is a finite list of capital letters : A, B, or C.

Uncertain state UCS = { ((A B C), 0.15), ((A C B), 0.30),
                        ((B C A), 0.30), ((C B A), 0.25) } .
```

Figure 11(a) State Space SS and Uncertain State UCS

```
Qlst = { ( ANY STATE, { (A,A,0.88), (A,B,0.12)
                        (B,A,0.10), (B,B,0.85), (B,C,0.05)
                        (C,A,0.02), (C,B,0.03), (C,C,0.95) } ) }
```

Figure 11(b) Imperfect ICO Qlst that Retrieves the First Element of a List

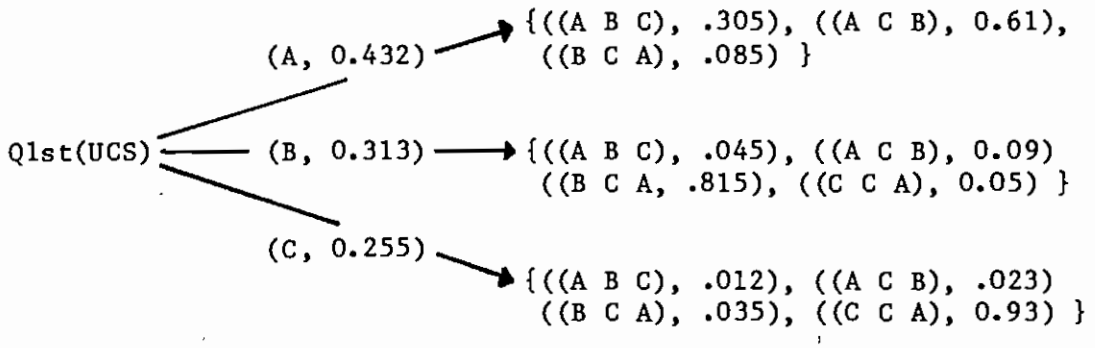

Figure 11(c) Applying Qlst to UCS

Figure 11 Uncertain List Processing with Imperfect Information Gathering

```
QXY(X,Y) = { ([(AT B, X, Y) or
               (NEAR B, X, Y)], {((X,Y),(X,Y), 1.0)} )

            (OTHERWISE
                (AT B, x, y),   {((x,y), (x,y), 0.20)
                                 ((x-1,y), (x,y), 0.20)
                                 ((x+1,y), (x,y), 0.20)
                                 ((x,y-1), (x,y), 0.20)
                                 ((x,y+1), (x,y), 0.20) } )  }
```

Figure 12(a) Specification of Directionally Sensitive Operator QXY(X,Y)

```
UCS = { ((NEAR B, 20, 20), 0.40), ((AT B, 30, 30), 0.60) }
```

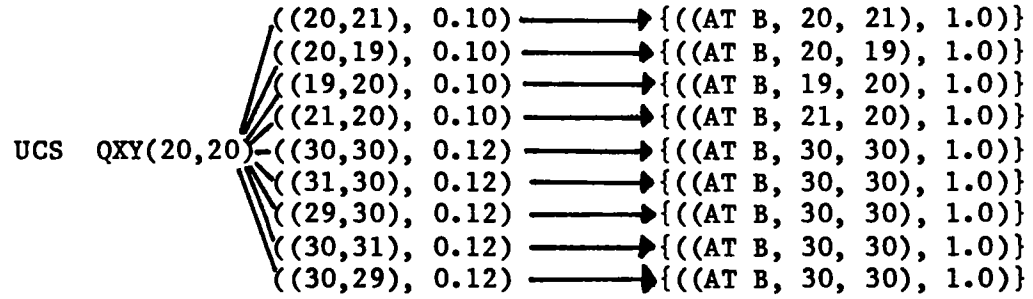

Figure 12(b) Application if QXY(X,Y) to an Uncertain State

Figure 12 An Imperfect IGO for the Uncertain Robotics Problem Space

Figure 12 presents an imperfect IGO applicable in the table top, robotics problem space defined in Figure 4. The operator QXY(X,Y) is

meant to model a directable location sensor pointed at coordinates (X,Y). If the block is at or near that location, the IGO returns the correct location as its result. If it is not, QXY(X,Y) can locate the block only at positions at or near its true location, with all such results being equally likely. Figure 12(b) demonstrates application of the operator to a given uncertain state. Of interest in the example is that this one application of the operator is sufficient to exactly determine the position of the block; the outcome states for all five results at or near (30,30) have (AT B, 30, 30) as their only consistent component state. These equivalent outcomes could be collapsed to one uncertain state. There will be more on this topic below as we turn our attention to methods of problem solving with information-gathering operators.

VII PROBLEM SOLVING WITH INFORMATION GATHERING

We would like to describe state space search methods for uncertain problem solving that allow incorporation of information-gathering operators (IGOs) into solution plans. When it contains IGOs, a solution plan is no longer a simple sequence (of states and operators), but is rather a tree originating at ucs (i.e., having ucs as its root) and ending at a number of leaf states. Tree-structured plans present several new difficulties that must be dealt with during forward-directed search for a problem solution. One is how to extract a best plan, in terms of degree of goal space satisfaction, starting at ucs from the search tree that exists at any point during problem solving. In general, this capability is required to decide whether an uncertain problem has already been solved. A second is how to determine a best plan from ucs in the current search tree that includes a given uncertain state as a leaf. This would allow USOLVE to select an uncertain state in FRINGE, determine a best plan having that state as a leaf, and apply Step 3 as before.

Figure 13 presents a set of functions, headed by BESTPLAN, that provides the first capability. BESTPLAN(US) returns a pair of values: the degree of goal space satisfaction and the set of leaf states associated with a best plan within the current search tree having US as its root. A best plan is either US itself or a plan through successors of US. Successors are represented as a list whose elements are either individual uncertain states or, when an IGO has been applied, a sublist of consequent uncertain states. When a sublist is encountered, the degrees of satisfaction for leaf states of best plans starting at each element of the sublist must be combined. The degree of satisfaction contributed by a leaf state lf to an overall plan must be tempered by the probability that lf will be reached; thus, the contribution of lf is

equal to uprob(lf)*dsat(lf). The degree of satisfaction of a plan is simply the sum of the contributions of its leaves. A solution to an uncertain problem has been realized when the degree of satisfaction of BESTPLAN(ucs) >= ST. If a minimum-expected-cost best plan is desired, all that function BESTSUCPLAN needs to do additionally is to compare the expected costs of equally satisfying successor plans. BESTPLAN already prefers a plan with a given uncertain state as a leaf if all plans through its successors do not improve the degree of goal space satisfaction.

Figure 14 presents function BESTLEAFPLAN, which determines a best plan within the current search tree originating at ucs and having a given uncertain-state LF as a leaf. Note that the backward search for such a plan can halt when all information-gathering operators between LF and ucs have been encountered (i.e., the probability of the considered uncertain state is 1.0). BESTLEAFPLAN uses BESTPLAN and functions employed by BESTPLAN. With BESTLEAFPLAN applied to the selected uncertain state in Step 3, USOLVE has been generalized to accommodate information gathering. BESTLEAFPLAN not only returns a leaf set determining a best plan, but the plan's degree of goal state satisfaction as well. This degree of satisfaction can be compared with ST; if above the threshold, the leaf set can be used to retrieve the solution plan.

IGOs do not directly improve the degree of satisfaction of a best plan, but rather map a single uncertain state into several new uncertain states. As such, USOLVE would do well to limit their application to those uncertain states having significantly high levels of disunity. The threshold for applying IGOs would depend upon the solution threshold ST in an indirect manner (i.e., the higher the ST, the lower the disunity threshold for applying IGOs. USOLVE could also restrict the number of IGOs applied to a selected uncertain state or incorporate into the search tree only those that significantly reduce the sum or maximum of the disunity levels of consequent uncertain states. Optimal IGOs for a given uncertain state can be defined as those that minimize either the

```
function BESTPLAN(US)
  begin
    Set usplan to (satisfaction: dsat(US) * uprob(US),
                   leaves:   { US } );
    If [successors(US) = NIL] then Return(usplan);
    Set sucplan to BESTSUCPLAN(successors(US));
    If [satisfaction(sucplan) > satisfaction(usplan)]
        then Return(sucplan)
        else Return(usplan)
  end

function BESTSUCPLAN(SLST)
  begin
    Set bstplan to EVALPLAN(first(SLST));
    Set SLST to rest(SLST);
    While [SLST <> NIL] do
     begin
       Set nxtplan to EVALPLAN(first(SLST));
       If [satisfaction(nxtplan) > satisfaction(bstplan)]
        then set bstplan to nxtplan;
       Set SLST to rest(SLST)
     end;
    Return(bstplan)
  end

function EVALPLAN(SUC)
  begin
    If [SUC is a list]
        then Return(EVALIGOPLAN(SUC))
        else Return(BESTPLAN(SUC))
  end

function EVALIGOPLAN(SUCS)
  begin
    Set tplan to (satisfaction: 0, leaves: {});
    While [SUCS <> NIL] do
     begin
       Set nplan to BESTPLAN(first(SUCS));
       Set tplan to COMBINE(tplan, nplan);
       Set SUCS to rest(SUCS)
     end;
    Return(tplan)
  end

function COMBINE(PL1, PL2)
  begin
    Return(satisfaction: satisfaction(PL1) + satisfaction(PL2),
           leaves: leaves(PL1) + leaves(PL2))
  end
```

Figure 13 Functions Determining a Best Plan from an Uncertain State US

```
function BESTLEAFPLAN(LF)
  begin
    Set bstlfplan to (satisfaction: dsat(LF) * uprob(LF),
                      leaves: { LF } )
    While [ uprob(LF) < 1.0 ] do
     begin
      If [preop(LF) is an IGO] then
       begin
         Set others to list of other states produced by preop(LF);
         Set bstlfplan to COMBINE(EVALIGOPLAN(others), bstlfplan)
       end;
      Set LF to pred(LF)
     end;
    Return(bstlfplan)
  end
```

Figure 14 A Function Determining a Best Plan Containing Given Leaf LF

sum or the maximum of disunity levels associated with consequent uncertain states. This approach to controlling the application of IGOs eschews the notion that, after each unreliable operator is applied, as much sensing as possible is done to best assess the resultant situation. Rather, the approach described here tries to limit information gathering to that necessary to realize an adequate solution plan.

In the forward-directed search generated by USOLVE, information gathering can be marshaled to control disunity; pragmatic focusing would be primarily fortuitous. What is needed is a backward-directed search component generating precondition states from which a goal state can be reached with probability greater than ST. We refer to such states as the set of pragmatic states. Sequences of one or more IGOs could be applied to a selected uncertain state by USOLVE to determine whether a significant percentage satisfies components of the set of pragmatic states. Those contingencies can be considered solved, and eliminated from further consideration. Note that USOLVE could do this without a backward-search component, the set of pragmatic states reducing to GS. Figure 15 presents a function PRAGSTATES, which generates a subset of the pragmatic states for a given UPS. PRAGSTATES returns only a subset as a given set of effects may map into more than one pragmatic state,

thus increasing the pragmatic probability of its precondition. With each pragmatic state us we associate a pragmatic probability pprob(us), a successor psuc(us) and successor operator psucop(us). The latter two can be used to retrieve a sufficiently reliable plan from that state when encountered during problem solving. USOLVE could employ PRAGSTATES as an initial step, the resultant backward search being bounded by interaction between ST and operator reliabilities.

```
function PRAGSTATES
  begin
    Set pstates to { }.
    Set nstates to GS.
    For each s in nstates, set pprob(s) to 1.0.
    While [ nstates <> NIL ] do
     begin
       Set us to element of nstates.
       Set pstates to pstates + us.
       For each operator uo with ith precondition and jth effect
                such that us is consistent with eff(uo,i,j)
         begin
           Set ns to (a regressed copy of) pre(uo,i);
           Set pprob(ns) to peff(uo,i,j) * pprob(us);
           If [ pprob(ns) >= ST ] then
            begin
              Set psuc(ns) to us;
              Set psucop(ns) to uo;
              Set nstates to nstates + ns
            end
         end
       Set nstates to nstates - us
     end;
    Return(pstates)
  end
```

Figure 15 A Function Determining the Set of Pragmatic States for a (Globally) Given Uncertain Problem Space

Several heuristics for controlling problem-solving search by USOLVE have been mentioned. The equivalence of several uncertain states associated with the different results of applying QXY(X,Y) in Figure 12 suggests a final heuristic. Within a given search tree, similar

uncertain states may occur. Let the measure of similarity of two uncertain states usl and us2, sim(usl, us2), equal 1.0 minus one-half the sum of the absolute differences between probabilities of equivalent components in usl and us2. Thus, identical uncertain states have a similarity measure of 1.0, while totally disjoint uncertain states have a similarity of 0.0. A plan that is highly successful from one uncertain state may be expected to be somewhat successful from other, similar uncertain states. The indexing of uncertain states according to their most likely components could make the use of such a heuristic feasible. With each uncertain state we can associate a best plan in the current search tree. Such best plans can be maintained by a partial traversal of uncertain states on the path back toward ucs whenever a new leaf is generated during search. Then, when an uncertain state is selected for development, best plans from similar states can be considered as well as arbitrary operators.

VIII APPLICATIONS

In this section we briefly outline three planning issues that can benefit from application of our model. These are incremental planning, planning in conflict situations, and planning with abstract operators.

The notion of a solution in uncertain problem solving is a plan that will produce a goal state with probability above a specified solution threshold. In plans that employ information-gathering operators, we have seen that this solution probability is not required from each consequent state of an IGO. Therefore, during execution of a solution plan, a problem solver may find itself at a consequent state that happens to be a contingency for which an adequate plan has not been developed. This situation prompts consideration of replanning from the current uncertain state. More formally stated, an incremental planning system can follow this scenario: first, it determines a solution from ucs, associating with each uncertain state us of the plan the degree of satisfaction bestsat(us) of a bestplan from us; during plan execution, whenever it arrives at a us such that bestsat(us) < ST, it replans by solving the problem again with us as ucs. According to this scenario, the problem solver is always on a path to an uncertain leaf state ulf having dsat(ulf) >= ST. Thus, when a leaf is reached and plan execution is completed, the probability that the current state is a goal state is greater than or equal to ST.

The above scenario forces a problem-solving system to replan only when it encounters a relatively unexpected contingency during plan execution. Feldman and Sproull [8] discuss a similar perspective upon real-world problem solving and offer a utility-based approach to the control of planning. In their model, probabilities are not directly incorporated into state and operator descriptions, but are represented as parameters of the context. Incremental planning is also a central

notion behind recent, goal-based approaches to computer chess [25]. Conflict situations, including game playing, appear to be natural domains for application of uncertain problem solving. The opponent move generator can be modeled naturally in terms of unreliable operators. The opponent has several possible move options, with selection probabilities made dependent upon the prevailing game situation. In most board games, a perfect IGO can be applied with respect to the current board configuration after each move. This view of an opponent yields a probabilistic game tree for analysis[2]. In most other conflict situations, ranging from games like bridge and poker to business and military confrontations, imperfect IGOs and uncertain states are likewise prevalent [9]. If we assume that goal spaces and solution thresholds for such contexts can be adequately specified, the application of problem solving in uncertain problem spaces would be appropriate.

Finally, research in artificial intelligence has shown the utility of constructing plans at several levels of abstraction, both for efficiency in planning and for comprehensibility of resultant plans [18,19]. An abstract operator is one that has several possible implementations in terms of actions within the real-world context. For example, the acquisition of a tool may involve buying, borrowing, or simply retrieving it. As such, an otherwise reliable, abstract operator has uncertain side effects, depending upon which of the possible implementations is selected at execution time. Maintaining execution time flexibility in operator implementation can be important in dynamic, real-world contexts. In cooperative problem solving, in which one agent requests that another execute a given operator, uncertainty as to its implementation and to the resultant state, must be taken into account. Uncertainty can be controlled in this case by providing (rather than gathering) information to influence the other agent's choice of implementation.

We can model abstract operators by a mixed (reliable and unreliable) representation. Each abstract operator will have an intent

and a side effect aspect. The intent is modeled as a reliable operator with required preconditions and guaranteed effects. The side effect aspect is modeled as an unreliable operator with precondition sets that partition the states satisfying preconditions of the operator's intent; for each precondition of the side effect aspect, there is a set of possible side effects with associated probabilities reflecting the likelihood of corresponding implementations. To solve a problem, one first solves it in the certain terms of operator intents, merely carrying along side effect probabilities. Then one searches for conflicts between side effects and the preconditions necessitated by subsequent operator intents. Reference to a specified solution threshold determines whether detected conflicts can be ignored or a plan must be revised, possibly by the introduction of information-gathering operators.

IX CONCLUSION

As noted in the introduction, the artificial intelligence research community has recently begun to consider a variety of issues associated with real-world problem solving. These have included the coordination of multiple-goal satisfaction [6,15], cleaning up [6], cooperative (distributed) problem solving [3,23], and even waiting as a productive problem-solving activity [7].

We have examined an issue here that underlies all aspects of real-world problem solving, i.e., that of dealing with uncertain states and unreliable operators. We have described techniques for solving problems in lights of such entities. Our methodology provides for the application of information-gathering operators within plans. The use of these operators yields tree-structured plans; algorithms are presented that search for such plans. Several potentially useful heuristics are described.

Our model is based on a probabilistic generalization of the notion of problem space. However, probabilistic representations of uncertainty do have their drawbacks [20]. For one, there is no adequate way to represent ignorance; maximum entropy approaches do not truly capture the notion of lack of knowledge. Other representations of uncertainty, such as fuzzy sets [26] and support-plausibility intervals [10,20], offer alternative approaches to modeling the uncertainty inherent in real-world problem solving. The implementation and evaluation of these approaches constitute a currently active area of artificial intelligence research.

REFERENCES

1. R. B. Banerji, Artificial Intelligence: A Theoretical Approach. New York, New York: North-Holland Publishing Company, 1980.

2. A. W. Burks, Chance, Cause, Reason. Chicago, Illinois: University of Chicago Press, 1977.

3. D. D. Corkill, "Hierarchical planning in a distributed environment," in Proc. Int. Joint Conf. on Artificial Intell., Tokyo, Japan, pp. 168-175, August 1979.

4. J. H. Eaton and L. A. Zadeh, "Optimal pursuit stategies in discrete-state probabilistic systems," Journal of Basic Engineering, pp. 23-29, 1962.

5. Y. M. El-Fattah, "Recursive algorithms for adaptive control of finite Markov chains," IEEE Trans. on Systems, Man, and Cybernetics, pp. 135-144, 1981.

6. A. M. Farley, "Issues in knowledge-based problem solving," IEEE Trans. on Systems, Man, and Cybernetics, pp. 446-459, 1980.

7. A. M. Farley, "On waiting," in Proc. First Annual Nat. Conf. on Artificial Intell., Stanford University, Stanford, California, pp. 128-130, August 1980.

8. J. A. Feldman and R. F. Sproull, "Decision theory and artificial intelligence II: the hungry monkey," Cognitive Science, pp. 158-192, 1977.

9. T. D. Garvey and M. A. Fischler, "The integration of multi-sensor data for threat assessment," in Proc. Fifth Int. Joint Conf. on Pattern Recognition, Miami, Florida, pp. 343-347, 1980.

10. T. D. Garvey, J. D. Lowrance, and M. A. Fischler, "An inference technique for integrating knowledge from disparate sources," in Proc. Int. Joint Con. on Artificial Intell., Vancouver, British Columbia, pp. 319-325, August 1981.

11. R. V. Houska, "Using matrix methods to find optimal controls for certain stochastic models," in Proc. Third Int. Joint Conf. on Mathematical Modeling, Los Angeles, California, July 1981.

12. R. A. Howard, Dynamic Programming and Markov Processes. Cambridge, Massachusetts: MIT Press, 1960.

13. J. G. Kemeny and J. L. Snell, Finite Markov Chains. Princeton, New Jersey: Van Nostrand Publishing Company, 1960.

14. D. E. Knuth, The Art of Computer Programming: Volume 3, Reading. Massachusetts: Addison-Wesley Publishing Company, p. 396, 1973.

15. D. McDermott, "Planning and Acting," Cognitive Science 2, pp. 71-110, 1978.

16. A. Newell and H. A. Simon, Human Problem Solving. Englewood Cliffs, New Jersey: Prentice-Hall Publishing Company, 1973.

17. N. J. Nilsson, Principles of Artificial Intelligence. Palo Alto, California: Tioga Publishing Company, 1980.

18. E. D. Sacerdoti, "Planning in a hierarchy of abstraction spaces," Artificial Intelligence 5, pp. 115-135, 1974.

19. E. D. Sacerdoti, A Structure for Plans and Behavior. New York, New York: North-Holland Publishing Company, 1977.

20. C. Shafer, A Mathematical Theory of Evidence. Princeton, New Jersey: Princeton University Press, 1976.

21. C. E. Shannon, "A mathematical theory of communication," Bell System Technical Journal, 27, pp. 379-423; 623-656, 1948.

22. R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable Markov processes over a finite horizon," Operations Research, 21, pp. 1071-1085, 1973.

23. R. G. Smith, "The contract net protocol: high-level communication and control in a distributed problem solver," IEEE Trans. Computers C27, pp. 1104-1113, 1980.

24. A. Waibel, N. Krishman, and R. Reddy, "Minimizing computational cost for dynamic programming algorithms," Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, June 1981.

25. D. Wilkins, "Using patterns and plans in chess," Artificial Intelligence, 14, pp. 165-203, 1980.

26. L. A. Zadeh, "Fuzzy sets," Information and Control, 8, pp. 338-353, 1965.