STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

January 1972

A CAUSALITY REPRESENTATION FOR ENRICHED ROBOT TASK DOMAINS

by

Peter E. Hart
Nils J. Nilsson
Ann E. Robinson

Artificial Intelligence Center

Technical Note 62

SRI Project 1187

The research reported herein was sponsored by the Office of Naval Research under Contract N00014-71-C-0294 and NR-348-028.

Reprinted from Technical Report dated December 1971 to Office of Naval Research, Contract N00014-71-C-0294 (AD 734 140).

ABSTRACT

Existing robot projects in the field of artificial intelligence have concentrated on tasks wherein the robot must excite an action in order to change the current state of its environment. In this report we consider other families of tasks, such as those involving the inhibition of an action or the maintenance of the current state of its environment. In particular, we propose a representation for knowledge that a robot would need if it is to solve problems from such task domains.

I INTRODUCTION

The goal of the research described in this report is to propose and to investigate possible learning strategies for robot control. Such strategies would construct and execute complex sequences of behaviot in dynamically changing environments. The present work is closely related to much of the research on intelligent robots being carried out here at Stanford Research Institute and at other Artificial Intelligence centers in this country and abroad. The usual robot work, however, has focused on one special class of tasks: namely, tasks involving the execution of an action or a series of actions. We might call these tasks excitatory change tasks, since the robot must excite an action to change the current state of affairs. In our present work we consider many additional families of robot tasks, and we suggest a representation for the information a robot must have for successfully accomplishing tasks in all of these families.

II MODES OF CAUSALITY

A. Typical Mammalian Behavior

To motivate our discussion, let us consider families of behavior typically engaged in by humans as well as many lower animals. Among the families of behavior are the following:

(1) An organism will take a positive action to get a reward, such as pushing a lever to get food. A general term for this is excitatory reward, indicating that an action is excited in order to get a reward.

(2) A second type of behavior is to inhibit doing an action in order to avoid some bad consequence of that action. An example is not crossing a railroad track when a train is coming. This is called inhibitory avoidance behavior. Notice that inhibitory behavior implies that the organism has at least two concurrent goals calling for conflicting actions. In our present example, if the only goal were to

avoid a train, then there would would be no reason ever to cross a railroad track, and consequently never an action to inhibit.

(3) Excitatory avoidance is another type of behavior in which a positive action is taken to avoid the occurrence of an event. An example is jumping out of the way of an oncoming car to avoid getting hit.

(4) As a final example, organisms will take a positive action to escape a bad situation. To go inside a shelter out of a blizzard is to escape a storm. This is called excitatory escape.

B. Modeling Causality

The types of behavior listed above indicate that there are other families of robot tasks beside excitatory change tasks. Let us now systematically explore the possible families of robot tasks.

The basic elements in our discussion are variables named R, Q, and a. R and Q are predicates about the state of some things in the world. For the present, assume that the range of their values is [T,F]. When we say that "R flips" we mean that it goes either from false to true, or from true to false. We say that a is an action that can be taken in the situation defined whenever Q has the proper value.

We can discuss the probability that action a will cause R to change its truth value and also the probability that a will prevent R from changing its value. There are four general cases to consider:

(1) Prob[$Q \xrightarrow{a}$ R flips]: This probability, which we name p, denotes the probability that action a, taken when Q has the proper value, causes R to change its truth value. For example, this is the probability that (a) if a person is at the light switch (predicate Q) and (b) if he flips the switch (action a), then (c) the light will change states (predicate R). Note that, for now, we are not concerned with whether the light was initially on or off but only that it will change.

(2) Prob[$Q \xrightarrow{a}$ - (R flips)]: This denotes the probability that action a, taken when Q has the proper value, prevents R from changing. To continue our example, this is the probability that the light will not change when the light switch is flipped. For the same Q, R, a, and p from (1), the probability is 1 - p.

(3) Prob[$Q \xrightarrow{-a}$ (R flips)]: This probability, which we name q, denotes the probability that action a, not taken when Q has the proper value, causes R to change. Using the same example, this is the probability that not flipping the light switch when the person is next to it will cause the light to change states.

(4) Prob[$Q \xrightarrow{-a}$ - (R flips)]: This denotes the probability that action a, not taken when Q has the proper value, prevents R from changing. Using our example, this is the probability that not flipping the light switch when the person is next to it will lead to the light not changing. This is exactly the probability 1 - q.

Rewriting these four probabilities using the notation of conditional probability, we can see that p and q are distinct probabilities because they are conditioned on different events. It is also evident from the four new formulas below that if p = q then whether R flips states is independent of the action a. The four formulas in our alternative notation are:

(1) Prob[R flips | Q, a] = p

(2) Prob[- (R flips) | Q, a] = 1 - p

(3) Prob[R flips | Q, -a] = q

(4) Prob[- (R flips) | Q, -a] = 1 - q

Note that we have not yet said whether predicate R is changing from false to true or from true to false, nor have we said whether it is initially false or true. These cases will be elaborated shortly.

C. Behavior Based on Causality

The probabilities discussed above specify the effects on predicate R of executing or not executing an action a in the presence of a second predicate Q. Based on these probabilities, we can enumerate eight types of behavior that a robot may engage in for the purpose of creating or maintaining an acceptable world situation. These eight types are obtained by considering three independent factors: (1) whether the maximum of p and q is p or q; (2) whether we want to change or maintain the value of R; and (3) whether R is initially true or false. The probabilities p and q can both range over the unit interval and so take on an infinite number of possible values. We saw earlier that if p = q then the action is irrelevant; we will regard this as a degenerate case. What is critically important is which of the p and q is greater, for this determines whether exciting or inhibiting an action is more likely to affect R. For simplicity, we will say that when p is maximum, p = 1 and q = 0, and when q is maximum p = 0 and q = 1. The eight cases are given in Table 1.

Table 1

p q	Change/Maintain Predicate R	Initial Value of R → Desired Value of R
1 0	Change	F → T T → F
	Maintain	T → T F → F
0 1	Change	F → T T → F
	Maintain	T → T F → F

III THE REPRESENTATION

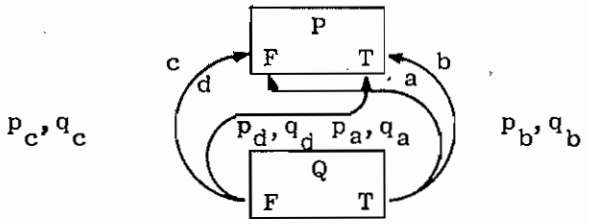
To model a robot's total knowledge of causality, we may construct a net whose elementary components model the basic causality rules. To simplify our discussion, we will assume that the robot can at all times evaluate any predicate without having to perform information-acquiring acts. This restriction is of course unrealistically severe, and its relaxation constitutes an important subject of future research.

A. Primitives

We assume predicates and actions that are consistent, at least in spirit, with the top-level primitive actions and predicates used in the Stanford Research Institute robot system. A typical predicate might specify the location of the robot or other objects in the environment, whether a door is open, and so forth. A typical action might be GOTO(), where the argument (i.e., the goal location of the robot) must be specified by the top-level robot problem-solver. We might also envision a set of "internal" predicates that tell the robot about its own state of being: whether its batteries are charged, whether it needs repairs, and so forth.

B. The Net Representation

We suggest a net representation of causality. The nodes of the net correspond to predicates about the primitive stimuli in the world. Each node has a "false" and a "true" side. Arrows whose tails are attached to a particular side of a node correspond to the possible actions that can be taken when the predicate has that value. Heads of arrows indicate consequences of actions, as illustrated by the following example showing all the possible connections for two nodes in the tree:



Each arrow from Q to P is labeled with an action and associated probabilities p and q. For example, the arrow labeled with action a above means that when Q is true and P is false then p_a is the probability that action a will cause P to flip from false to true and q_a is the probability that not doing a will cause P to flip. If $p_a = 1$ and $q_a = 0$ then P will always flip from false to true if a is done and it will never flip if a is not done. Note that this does not mean that nothing else will change P. It may be that there is some other way of changing P, but the new way does not depend on Q. If $q = 1$ and $p_a = 0$ then when Q is true a will always prevent P from flipping from false to true, but unless a is done P will flip. We can summarize this state of affairs as follows: Given that Q is true, $p_a = 1$ and $q_a = 0$ means that R flips if and only if action a is executed. Conversely, $p_a = 0$ and $q_a = 1$ means that R flips if and only if action a is not executed. A similar explanation holds for the other three arrows.

We use the term enabled to describe the arrows whose tails come out of the side of the node corresponding to the current value of the node. In the above example, if Q is true then arrows a and b are enabled. Relevant arrows are those whose heads come into the side of the node which corresponds to the current value of the predicate. If P is true above, then arrows b and d are relevant.

Each node contains two kinds of information: the value of the predicate and the urgency attached to maintaining or changing that value. The predicate values are written v(Pred) and lie on the interval -1 = false to +1 = true. For now assume that there are only two values, ±1, and later we will see why they become analog.

The urgency of a node, u(Pred) also lies in the interval of +1 to -1. An urgency of +1 means that the predicate must be made or kept true, and an urgency of -1 means that the predicate must be made or

kept false. An urgency of 0 means "don't care." The urgency of a node is a function of the urgency of its predecessor, the value of its predecessor, how probable it is that an action will change or prevent something (the p and q values), and other factors such as the effort needed to take the action.

The nodes described above are to be connected into a network from which a robot computes its actions. At present, all of our networks have been trees, and we do not know whether this will change. Figure 1 shows a simple illustrative network consisting of two subtrees, each subtree corresponding to one top-level goal.

IV PROCESSING THE REPRESENTATION

We have developed a general algorithm whose purpose is to compute an appropriate action to take based on the trees of predicates and actions. Once an action is determined and executed the trees are searched again to find a new action. The search starts at the top nodes, whose urgencies are assumed known, and looks for the most urgent enabled action. This is done by searching the nodes according to their computed urgencies.

Before giving a detailed statement of the algorithm, we will sketch its operation on subtree A of Figure 1:

(1) At node 1 compare the value and the urgency. If v(1) = u(1) then STOP. Else,

(2) Compute the successors of 1 by following the relevant arrows backward out of 1. In this case the successors are 2 and 3.

(3) Look for the most urgent tip node (or open node). If max[u(2),u(3)] is enabled, then execute the action associated with it. For example, if u(2) > u(3) and v(2) (at door) is true, then the appropriate action is "open the door." Note that the arrow from node 3 to node 1 has no action on it. If v(3) is true, then the person may open the door (with probability = 0.5) and the robot need do nothing.

(4) If u(2) > u(3) and node 2 is not enabled (i.e., the robot is not at the door), then continue the search below 2.

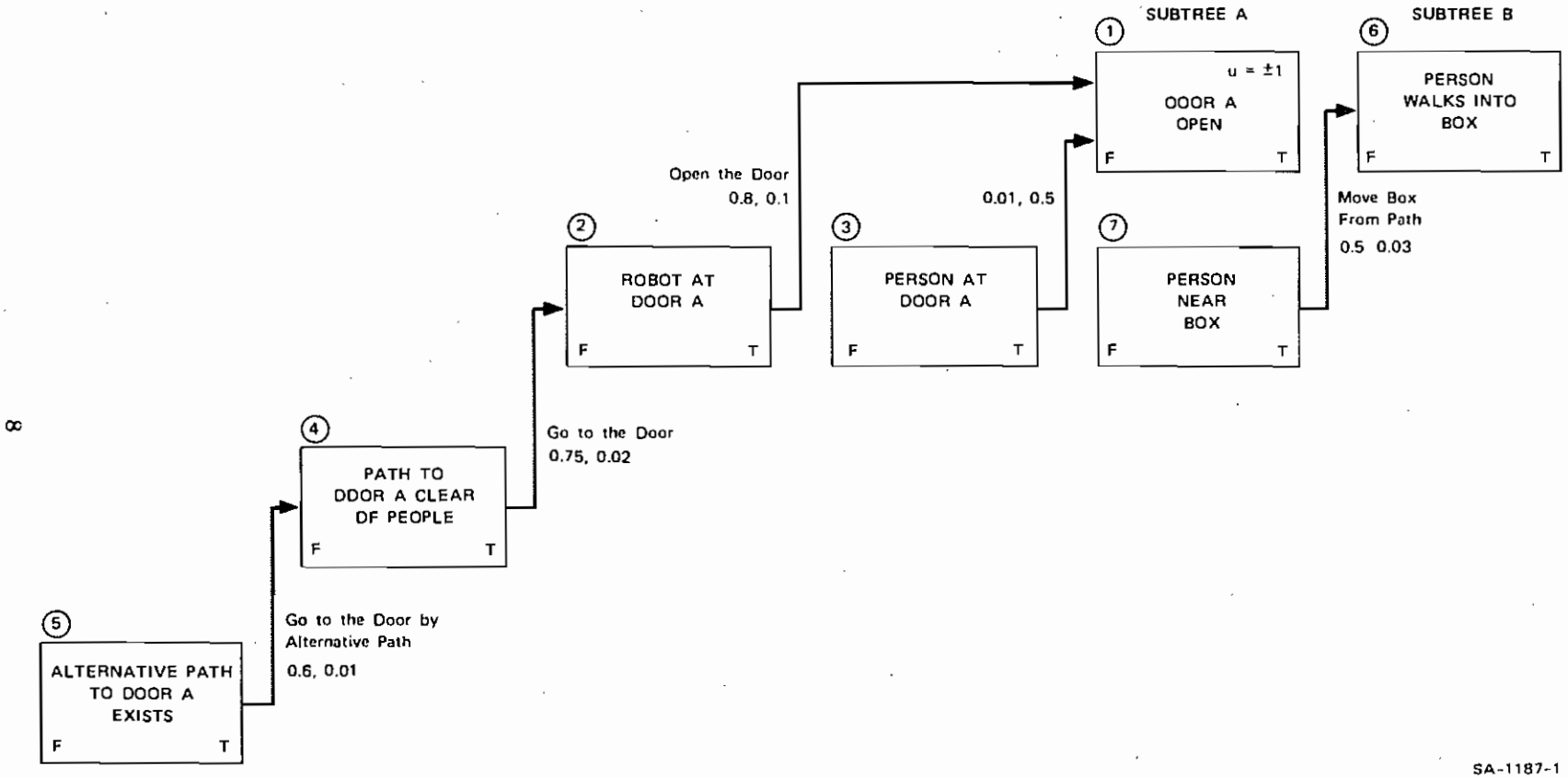
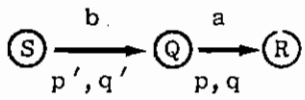


FIGURE 1 ILLUSTRATIVE NETWORK

V THE BASIC ALGORITHM

The basic search algorithm selects the most urgent node in the tree and takes appropriate action. For illustrative purposes, consider the net fragment shown below, in which Q is the most urgent open node:



For any open node, there are eight cases to consider. These eight cases are based on the following three dichotomies:

(1) $\text{sgn}[u(R)] = \text{sgn}[v(R)]$ or $\text{sgn}[u(R)] \neq \text{sgn}[v(R)]$. This tells us whether we want to maintain the value of R of flip it. If the signs are equal, then we want to maintain it; otherwise we want to flip it.

(2) $p > q$ or $p < q$. This tells whether an action should be done to maintain R or to flip R. If $p > q$ then R is more likely to change if a is done. If we want to flip R we do a, and if we want to maintain R then we do not do a. Similarly, if $q > p$ we will do a to prevent the flip of R if that is desired, and not do a if we want R to flip.

(3) Action a is disabled or enabled. If a is enabled we can do it. If a is disabled we may try to change Q to enable it.

For each combination of these three situations the algorithm either selects an action to take or continues searching the tree. Some examples of cases are:

(1) $\text{sgn}[u(R)] \neq \text{sgn}[r(R)]$

$p > q$

a is enabled.

This is a common case of excitatory change, and unless there is some reason to inhibit a (discussed below), we will execute it.

(2) $\text{sgn}[u(R)] \neq \text{sgn}[v(R)]$

$p > q$

a is disabled.

We want to do a so we will try to enable it. To do this, compute the relevant successors of Q, say S. We then try to compute u(S) and in particular set $\text{sgn}[u(S)]$ to the sign of the tail of the arrow from S to Q. This indicates that we want v(S) to be such that b is enabled. Then when b is done, Q should flip enabling a.

(3) $\text{sgn}[u(R)] = \text{sgn}[v(R)]$

$p > q$

a is enabled.

This is a case of either inhibitory maintenance because R is as we want it now and we discover that doing a will cause R to change. In this case, we put a on a "candidate inhibition list." Later, if we want to do action a for some other reason, we can decide whether it is more urgent to do a and change R or not to do a and maintain the current value of R. The list is called a candidate list because when we add the possibility of imperfect knowledge, we may not know whether a is enabled, and it may be costly to find out until we have a good reason for doing action a.

(4) $\text{sgn}[u(R)] = \text{sgn}[v(R)]$

$q > p$

a is enabled.

This means that unless we do action a predicate R will change value. Since R is as we want it, we will execute a to prevent the change (subject to checking for any inhibitions on a). This is the case of excitatory maintenance.

VI ELABORATIONS

There are a number of ways to complicate this model; in the following we limit ourselves to discussing two possibilities. These two are the "knowledge" problem and the problem of learning.

A. Knowledge

It is quite unrealistic to assume that a robot at all times has perfect knowledge of his environment; it cannot and does not watch everything at once. Instead, attention is focused on those things which are currently of interest. We intend to incorporate into our model a system whereby the robot will "forget" things or begin to mistrust its memory, and thereby force it to look again to find the current status of relevant stimuli. To do this we expect to have a data base of the robot's current information about the state of the world. These data may not reflect the true state of the world reliably, but it is not unrealistic for a robot to have imperfect knowledge. The nodes of the tree will be evaluated from the model.

The form of storage for the data base is still to be determined, but we have several possibilities. One requirement is that there should be some concept of time in the world: What occurred after what? How long ago did something change? How often does it change? All of these are in various ways dependent on time. This suggests that we store information in the form of a time line. Each observation about the world and each action taken can be stored on this time line. We also may use some idea of semantic storage such as those studies by Quillian, Becker, and others.

Another problem to be considered is that things can be forgotten that were once known while remembered "facts" may no longer be true. For this reason we may sometimes not trust our knowledge. A third thing that affects our knowledge of the world is the actions that we take. We may do something that we think or know affects certain items. After that action is taken we may not know the new state of the affected items until we check them. (We really do not know whether the light

switch worked until we see the light go on or off. If the light is in another room we may want to go see whether the light is on before we claim it is.) Because of these imperfections in our knowledge we want the nodes of the net to have analog truth values. These analog values tend toward 0 as a fact becomes unknown and they tend toward ±1 true or false when we check for the current state of that fact.

To reflect in our net the possibility that we might have to take an action to learn the value of a node, we have added knowledge acquisition arrows which point to the middle of the node. Whenever the predicate in the node is at or near 0 the relevant arrow is the acquisition arrow and it will be searched for the action to take to acquire the knowledge desired. The effect of this action is to move the value of the predicate off center in either the false or true direction. The same search rules stated before can easily be generalized to handle this third type of arrow. The conditions for acquisition of knowledge can be as elaborate as desired. For instance, it could be that in order to know whether the light is on, one must go through several rooms to see it. Depending on how urgent the information is, the importance of doing the acquisition can be determined. Initially, we will not make knowledge acquisition too complicated, but it will be possible to make information-acquiring sequences quite elaborate.

B. Learning

The second major topic that we have begun to consider, and want to study in much more detail, is the problem of learning. The first and easiest things to learn are the probabilities (p and q) associated with already established nodes. Variation of p and q can have great effects on the actions taken because they help determine the order in which the tree is searched. Establishing the reliability of p and q seems straightforward. We want to base these figures on the prior experiences stored in the data base, and estimate the probabilities based on these "memories." With proper weights on the past experiences it should be possible to show changes in actions based on changing reinforcement schedules, and also to show the extinction of actions based on discontinuing reinforcement.

A second more interesting kind of learning is to invent new nodes and arrows. In this case we will have known actions that can be taken, some of which will produce desired results under the right conditions. We have some ideas, admittedly sketchy, about how the linking of a new node can happen. A possible approach is that whenever there is something to be done, such as getting to a new room, and there is no known way of achieving it, we can have some sort of random set of actions occur until something changes. We hope that eventually either we will be in the new room or else we will be in a state from which it is possible to reach the new room. The last action that was done before the change can then tentatively be linked with the changed node. Determining the conditions under which this action worked is subtle, but a first guess could be that the last thing or last few things that changed prior to the actions were those that were needed. A link can be made, and the next time the situation is encountered the action will be tried sooner because of this link. If the action seems to work, "reinforcement" will cause the probabilities on the arrows to increase. If the initial action or conditions were wrong, then the next time this may not work and the probabilities will be lowered and something else tried. In this fashion, new chains of actions can be constructed.