DON'T BE STUPID

Dennis Kibler
Paul Morris

Information & Computer Science Department
University of California at Irvine

ABSTRACT

We are studying control knowledge for a general problem solver, named BLOCKHEAD. Currently the problem solver is based on negative heuristics, which has led to surprising efficiency in suitable domains. The lesson seems to be that it is easier to avoid being stupid than to try to be smart. Stupid plans are defined and a plan improvement method proposed. Analyses of stupid plans and failed plans suggest effective negative heuristics.

Introduction

We are investigating a general problem solver which incorporates control knowledge in the form of negative heuristics. The problem solver accepts problems represented relationally, as did STRIPS [1,2], WARPLAN [3], and NOAH [4]. Operators are represented as relational productions, following Vere [5]. An unusual feature is that preconditions must match <u>distinct</u> relationships in the current state. This allows a more natural description of operators. For example, in the BLOCKS world the operator to move a block X onto a block Y includes the preconditions clear(X) and clear(Y). Since these may not match the same item, we do not need the additional condition notequal(X,Y).

In contrast to the above mentioned problem solvers, BLOCKHEAD does not employ goal reduction: it is a forward-chaining system. Nevertheless, it is goal directed, as we shall see. Forward-chaining systems have been generally neglected in studies of problem solving, perhaps because goal direction has been (erroneously) regarded as synonymous with goal reduction. Note that forward systems avoid the difficult issue of integration of partial plans.

Both breadth first and depth first variants of the interpreter have been constructed. Each variant avoids repeating already visited states; this generally produces a finite search space.

This research was supported by the Naval Ocean Systems Center under grant N66001-80-O0377.

Negative Heuristics

The problem solver is guided by negative heuristics. These tell one what NOT to do. They are not intended to replace positive heuristics, but rather to complement them. Negative heuristics are so natural that people are generally unaware of them. In the blocks world, four negative heuristics surprised us by eliminating nearly all search. Informally, they are:

1. Don't add to a pile containing a block that needs to be moved.

2. Don't add to a pile containing a block that requires adding another block to it (unless you are

presently achieving that goal).

3. Don't move a block off an irrelevant pile.

4. Don't move a block onto an irrelevant pile.

A more formal statement of the first heuristic is, (capital letters stand for variables):

don't make on(X,Y) if for some W and Z, an unachieved goal conjunct is on(Z,W), and Y covers W.

Note that the heuristics relate the goal to the current state and anticipated action. The heuristics are designed to eliminate actions which clearly do not contribute to the goal. This provides a form of goal direction which can be highly effective. Using these heuristics, both the breadth first and depth first systems solved typical ten block problems with almost no consideration of fruitless paths. A full width search of a ten block world might generate over 30 million states, so we were somewhat amazed that these few guidelines produced such limited searches.

It is of interest that the depth first system usually required less search than its breadth first cousin. This occurred because, in the breadth first approach, equally good partial solutions mist all be explored. With the heuristics, most paths being explored did in fact lead to solutions, giving the depth first system an advantage with its "don't care" approach. However, the breadth first problem solver has the advantage of always generating a least cost solution (modulo the heuristics), The depth first problem solver sometimes generates non-optimal plans.

<u>Stupid Plans</u>

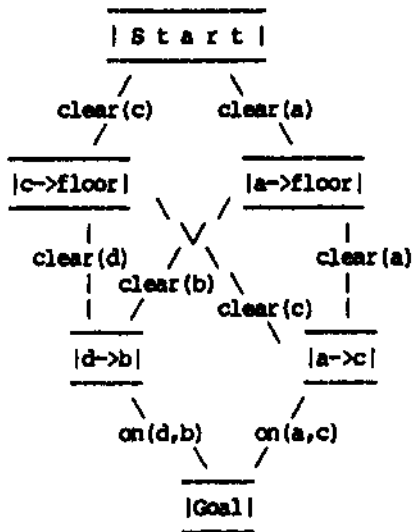
We say a plan is stupid if it contains a (proper) subplan that solves the problem. A plan which is not stupid need not be optimal. For example, suppose the initial state consists of four blocks, with b and d on the floor, a on b, and c on d. Let the goal be d on b and a on c, as pictured below.

```
initial state          goal
 | a |   | c |     | d |   | a |
 -----   -----     -----   -----
 | b |   | d |     | b |   | c |
```

A non-optimal plan to solve this problem is: c to floor; a to floor; d to b; a to c. This plan is not stupid because no subsequence of the plan will solve the problem. Each operation in the plan contributes to the solution. The plan can be improved by omitting the second operation and interchanging the last two operations. In our view this goes beyond plan simplification.

One could detect and simplify a plan by trying all possible subplans. Here we describe a less expensive algorithm which is not completely effective. Form a graph, the plan graph, in which each node corresponds to an operation in the plan and is labelled with that operation. A directed edge connects nodel to node2 if there are postconditions of nodel which are preconditions of node2 and are not preconditions of any intervening operations. The edge is labelled with this set of (instantiated) postconditions. Plan graphs are closely related to the triangle tables of Nilsson.

The plan given as an example would have the following plan graph. Nodes are enclosed in boxes, The arcs are directed downwards and are labelled with relationships.



Using a plan graph, we simplify a plan in the following way. Mark all the nodes that finally establish one of the goal conjuncts. All nodes that are not ancestors of any narked node can be deleted from the plan. Also, any directed path consisting of unmarked nodes that is reducible to the identity transformation can also be excised. This reduced plan graph can now be translated into a plan by choosing any total ordering of the operations which is consistent with the partial ordering determined by the plan graph.

negative Heuristic Discovery

Stupid plans and examples of poor search played an important role in our discovery of effective negative heuristics. When examining such failures, certain actions struck us as "stupid." Attempts to explain WHY they were stupid led to the negative heuristics. We believe this process can be mechanised. Work is proceeding on methods to identify the "point of stupid departure" and to generalize a description of the system state at that point.

Implementation notes: The breadth first problem solver was first programmed in LISP and then in PROLOG. The compiled PROLOG code executed four times faster than the compiled LISP code. The PROLOG code was more concise than the LISP code and easier to modify. If an AI problem requires pattern matching, non-determinism or execution of anchor trees, then PROLOG seems to have advantages over LISP.

REFERHNCES

[1] Nilsson, N.J., Principle of Artificial Tn»ninynffft. paio Alto: Tioga, 1980.

[2] Pikes, R.E. and Nilsson, N.J., STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligencer 1971, 189-203.

[3] Warren, D.H.D., WARPLAN: a system for generating plans, Memo 76, Dept. of Computation Logic, Univ. of Edinburgh, School of Artificial Intelligence, June 1977.

[4] sacerdoti, E.D., A Structure Plane and Behavior, New York: Elsevier, 1977.

[5] Vere, S.A., Relational Production systems, Artificial Intelligence, vol 8., 47-68.