Abstract
Recently, Knoblock has advocated a mechanism for automatically constructing planning hierarchies. We show that Knoblock’s method, and more generally, the principle of ordered monotonicity, can produce hierarchies that perform arbitrarily poorly. The reason is that Knoblock’s technique addresses only one of two important factors in ordering clauses. We propose a technique for the other based on evaluating the number of potential solutions to different possible subgoals in a plan.

1 Knoblock’s Method
Recently Knoblock [1, 2] has advocated a mechanism for automatically constructing fixed hierarchies to control planning search. The technique involves constructing a directed graph of potential conflicts between operators relevant to a goal. The graph is then broken into components and sorted to give the resulting hierarchy.

To illustrate Knoblock’s approach consider a simplified machine shop example with the following operators for shaping, drilling, and painting an object:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precs:</th>
<th>Effects:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shape(x)</td>
<td>Object(x)</td>
<td>Shaped(x), ~Drilled(x), ~Painted(x)</td>
</tr>
<tr>
<td>Drill(x)</td>
<td>Object(x)</td>
<td>Drilled(x), ~Painted(x)</td>
</tr>
<tr>
<td>Paint(x)</td>
<td>Object(x), Steel(x)</td>
<td>Painted(x)</td>
</tr>
</tbody>
</table>

Suppose that the goal is

\[ \text{Shaped}(x) \land \text{Drilled}(x) \land \text{Painted}(x) \]

The graph that Knoblock would construct for this problem is:

\[ \text{Shaped}(x) \rightarrow \text{Drilled}(x) \rightarrow \text{Painted}(x) \]

\[ \text{Object}(x) \rightarrow \text{Steel}(x) \]

According to Knoblock’s technique, the planner should work on \text{Shaped}(x) before \text{Drilled}(x) or \text{Painted}(x). After expanding \text{Shaped}(x), the planner should work on \text{Object}(x) (because it is unaffected by any operators). It should then work on \text{Drilled}(x) before \text{Painted}(x), and so on.

2 The Problem
Suppose that the initial conditions are such that there are 100 pieces of stock in the machine shop, but only one of them is made of steel. In this case, Knoblock’s approach would have the planner hunt through and build partial plans for many pieces of stock before finding one that is steel, and hence amenable to painting. In contrast, if the planner were to start work on \text{Painted}(x) followed by \text{Steel}(x), very little search would be required.

As this example illustrates, Knoblock’s technique can perform arbitrarily poorly in comparison to the optimal fixed hierarchy for a problem. More generally, Ordered Monotonic (OM) hierarchies [1, 3] have this unpleasant characteristic.
The problem is that there are two different reasons why a conjunctive goal may be more difficult to solve than the two conjuncts taken independently:

1. Action interference,
2. Variable binding conflicts.

Knoblock’s technique and OM attempt to address the first of these; they order clauses to minimize interference between actions. In fact, Knoblock’s technique imposes more ordering constraints than necessary to accomplish this task. In the example above, all possible operator conflicts can be resolved by simple temporal ordering constraints among the actions in the plan. These ordering constraints are detected and resolved by a non-linear planning system. This is discussed further in [6].

3 Evaluating Clause Difficulty

In our machine shop example, the primary difficulty is related to variable binding conflicts; i.e. finding a variable binding for $x$ that allows a solution to all three goal clauses. Knoblock’s technique and OM have nothing to say about this.

One approach to this problem is to estimate the number of possible solutions to each clause and order the clauses to minimize the size of the resulting search space. For the example above, it is relatively easy to see how this might be accomplished. For the clause Shaped($x$), there is only one possible operator that applies and its precondition Object($x$) has 100 different solutions. As a result, there are 100 possible solutions to Shaped($x$). Similarly, Drilled($x$) has 100 possible solutions. For the clause Painted($x$), only one possible operator applies, which has two preconditions Object($x$) and Steel($x$). Object($x$) has 100 solutions, but Steel($x$) has only one, so the conjunction has at most one solution. This means that Painted($x$) has at most one solution.

The clause Painted($x$) therefore has the fewest possible solutions. If the planner starts with that clause only one solution will be considered for the other two clauses and a minimal amount of search is done.

The possibility of recursion among the operators, adds additional complexity to the problem of calculating the number of solutions for clauses. Techniques for dealing with this are described in [6].

4 Conclusion

To control search in planning, we need a much better means of estimating the difficulty of solving the goals and subgoals in a planning problem. Knoblock’s technique and OM attempt to address one aspect of this problem; estimating action interference between subgoals. However, these techniques impose unnecessary and sometimes detrimental ordering constraints.

A second, and equally important aspect of problem difficulty is recognizing possible variable binding conflicts between goal clauses. Ordering clauses to minimize the size of the search space is a key to minimizing such conflicts. Doing this requires the ability to estimate the number of solutions possible for each different goal clause. We have given a hint as to how this might be accomplished and are currently implementing and evaluating this technique (see [6]).

Acknowledgments

Thanks to Craig Knoblock, Steve Minton, Qiang Yang and anonymous reviewers for comments and discussion. This work is supported by DARPA contract F30602-91-C-0031.

References