Complexity Results for Planning

Tom Bylander
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210
U. S. A.

Abstract

I describe several computational complexity results for planning, some of which identify tractable planning problems. The model of planning, called "propositional planning," is simple—conditions within operators are literals with no variables allowed. The different planning problems are defined by different restrictions on the preconditions and postconditions of operators. The main results are: Propositional planning is PSPACE-complete, even if operators are restricted to two positive (non-negated) preconditions and two postconditions, or if operators are restricted to one postcondition (with any number of preconditions). It is NP-complete if operators are restricted to positive postconditions, even if operators are restricted to one precondition and one positive postcondition. It is tractable in a few restricted cases, one of which is if each operator is restricted to positive preconditions and one postcondition. The blocks-world problem, slightly modified, is a subproblem of this restricted planning problem.

1 Introduction

If the relationship between intelligence and computation is taken seriously, then intelligence cannot be explained by intractable theories because no intelligent creature has the time to perform intractable computations. Nor can intractable theories provide any guarantees about the performance of engineered systems. Presumably, robots don't have the time to perform intractable computations either.

Of course, heuristic theories are a valid approach if partial or approximate solutions are acceptable. However, my purpose is not to consider the relative merits of heuristic theories and tractable theories. Instead, I shall focus on formulating tractable planning problems.

Planning is the reasoning task of finding a sequence of operators that achieve a goal from a given initial state. It is well-known that planning is intractable in general, and that several obstacles stand in the way [Chapman. 1987]. However, there are few results that provide clear dividing lines between tractable and in tractable planning. Below, I clarify a few of these dividing lines by analyzing the computational complexity of a planning problem and a variety of restricted versions, some of which are tractable.

The model of planning, called "propositional planning," is impoverished compared to working planners. It is intended to be a tool for theoretical analysis rather than programming convenience. Preconditions and postconditions of operators are limited to being literals, i.e., letters or their negations. An initial state then can be represented as a finite set of letters, indicating that the corresponding conditions are initially true, and that all other relevant conditions are initially false. A goal is represented by two sets of conditions, i.e., the goal is to make the first set of conditions true and the other set false. For convenience, these are called positive and negative goals, respectively. Operators in this model do not have any variables or indirect side effects.

Different planning problems can be defined by different constraints on the number and kind of pre- and postconditions. Figure 1 illustrates the results, showing which planning problems are PSPACE-complete, NP-hard (but in PSPACE), NP-complete, and polynomial.[1] These results can be summarized as follows:

> Propositional planning is PSPACE-complete even if each operator is limited to one postcondition (with any number of preconditions).
>
> Propositional planning is PSPACE-complete even if each operator is limited to two positive (non-negated) preconditions and two postconditions.
>
> It is NP-hard if each operator is restricted to one positive precondition and two postconditions.
>
> It is NP-complete if operators are restricted to positive postconditions, even if operators are restricted to one precondition and one positive postcondition.

[1] A problem is in PSPACE if it can be solved in polynomial space. As is customary, it is assumed that PSPACE-complete problems are harder than NP-complete problems, which in turn are harder than polynomial problems. However, even P = PSPACE is not yet proven.

This research has been supported in part by DARPA/ AFOSR contract F4962O-89-C-0110 and AFOSR grant 89-0250.

It is polynomial *if* each operator is restricted to positive preconditions and one postcondition. The blocks-world problem, slightly modified, is a subclass of this restricted planning problem.

It is polynomial if each operator has one precondition and if the number of goals is bounded by a constant,

It is polynomial if each operator is restricted *to* no preconditions.

One additional box in the figure identifies a commonality between four of the problems.[2]

The remainder of this paper is organized as follows. First, I describe previous results on the complexity of planning. Then, propositional planning is defined. Next, I demonstrate the complexity results and show how the blocks world is covered by one of the results. Finally, I discuss the impact of these results on the search for tractable planning.

2 Previous Results

The literature on planning is voluminous, and no attempt to properly survey the planning literature is attempted here. Instead, the reader is referred to Allen et al. [1990] and Hendler et al. [1990]. Despite the large literature, results on computational complexity are sparse. In turn, I discuss previous results from Dean and Boddy [19871, Korf [1987], and Chapman [1987].

Dean and Boddy [1987] analyze the complexity of temporal projection—given a partial ordering of events and causal rules triggered by events, determine what conditions must be true after each event. Their formalization of temporal projection shares many features with planning, e.g., their causal rules contain antecedent conditions (preconditions) and added and deleted conditions (postconditions). In fact, the notation for propositional planning is mostly borrowed from Dean and Boddy. However, they only consider problems of prediction in which a partial ordering of events is given, whereas the equivalent planning problem would be to find some ordering of any set of events that achieves some set of conditions.

Korf (1987) considers how various global properties of planning problems (e.g., serializable subgoals, operator decomposability, abstraction) affect the complexity of using problem space search to find plans. In contrast, J focus exclusively on local properties of operators. However, except for Korf s own analysis of operator decomposability [Korf, 1985], neither he nor I describe the relationship from these properties of planning problems to the properties of operators. Clearly, this is a "gap" that future work should address.

[2]The following are other results not judged to be as interesting, but arc included here for completeness. Propositional planning is NP-complete if each operator is restricted to positive preconditions and negative postconditions, even if restricted to one positive precondition and two negative postconditions. It is polynomial in the previous cases if the number of goals is bounded by a constant. It is polynomial if each operator is restricted to positive preconditions and positive postconditions.

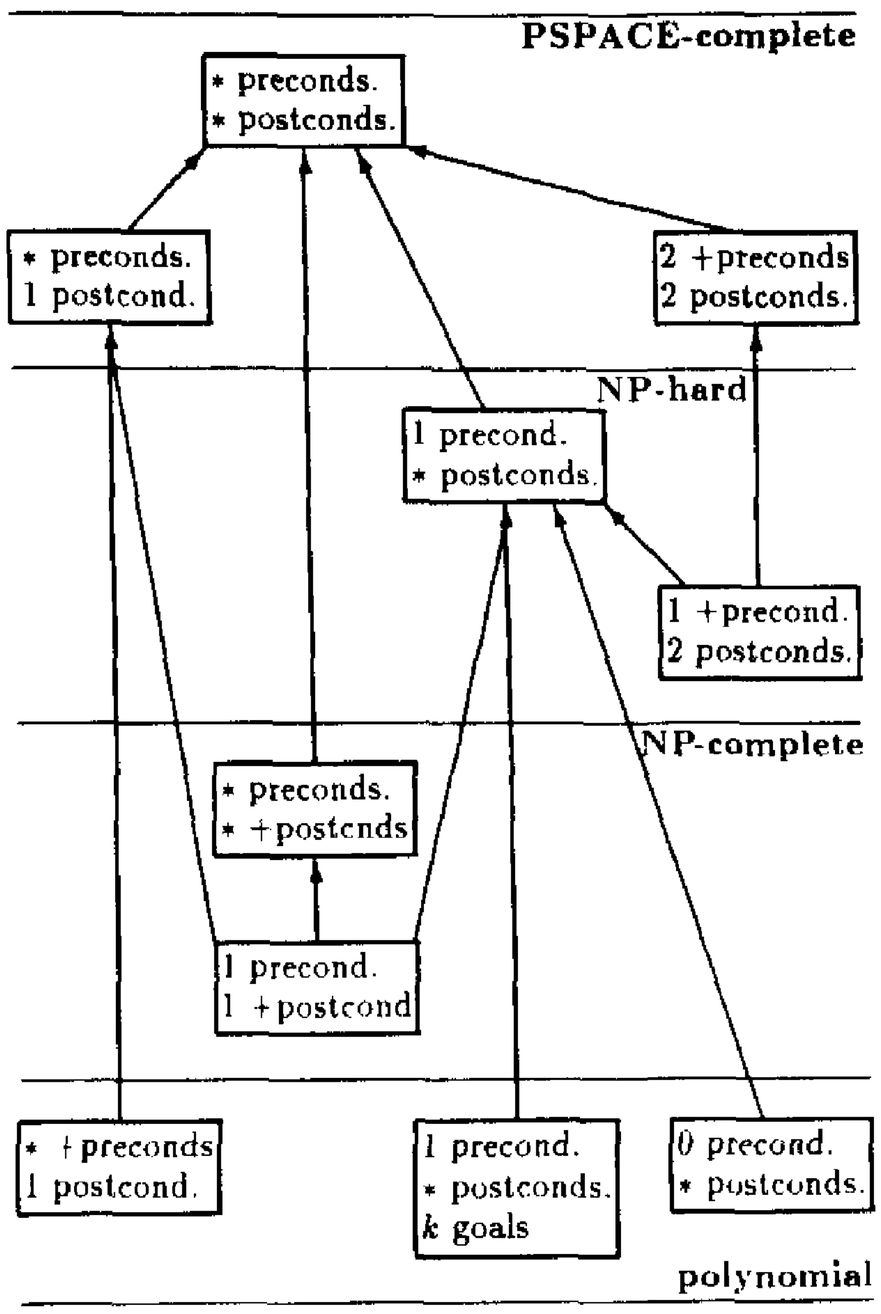

Figure 1; Complexity Results for Hierarchy of Propositional Planning Problems

Perhaps the most important complexity results for planning are due *to* Chapman's analysis of his planner TWEAK 'Chapman, 1987!. Because virtually all other planners are as expressive as TWEAK, his results have wide applicability, TWEAK'S representation includes the following features. The preconditions and postconditions of an operator schema are finite sets of "propositions," A proposition is represented by a tuple of elements, which may be constants or variables, and can be negated. A postcondition of an operator can contain variables *not* specified by any precondition of the operator, which in effect allows creation of new constants.

Chapman proved that planning is undecidable and so clearly demonstrated the difficulty of planning in general, but it is not obvious what features of TWEAK'S representation are to blame for the complexity. What happens to the complexity, for example, if postcoiidi* tions cannot introduce new variables? What happens if the size of states are bounded for any given instance of a planning problem? Are there any interesting restricted planning problems that are tractable? By considering a model of planning with considerably fewer features, the following analysis begins to address these questions.

3 Propositional Planning

An instance of *propositional planning* is specified by a tuple $(\mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G})$, where:

$\mathcal{P}$ is a finite set of *conditions*;

$\mathcal{O}$ is a finite set of *operators*, where each operator has the form $\langle \varphi, \eta, \alpha, \delta \rangle$:

$\varphi \subseteq \mathcal{P}$ is a set of *positive preconditions*;
$\eta \subseteq \mathcal{P}$ is a set of *negative preconditions*;
$\alpha \subseteq \mathcal{P}$ is a set of *positive postconditions*;
$\delta \subseteq \mathcal{P}$ is a set of *negative postconditions*; and
$\varphi \cap \eta = \emptyset$ and $\alpha \cap \delta = \emptyset$.

$\mathcal{I} \subseteq \mathcal{P}$ is the *initial state*; and

$\mathcal{G} = (\mathcal{M}, \mathcal{N})$ is the *goal*:

$\mathcal{M} \subseteq \mathcal{P}$ is a set of *positive goals*;
$\mathcal{N} \subseteq \mathcal{P}$ is a set of *negative goals*; and
$\mathcal{M} \cap \mathcal{N} = \emptyset$.

That is, $\mathcal{P}$ is the set of conditions that are relevant. Any state can be specified by a subset $S \subseteq \mathcal{P}$, indicating that $p \in \mathcal{P}$ is true in that state if and only if $p \in S$. $\mathcal{O}$ is the set of the operators that can change one state to another. $\mathcal{I}$ specifies what conditions are true and false in the initial state, i.e., $p \in \mathcal{P}$ is initially true if $p \in \mathcal{I}$ and initially false otherwise. $\mathcal{G}$ specifies the goals, i.e., $S \subseteq \mathcal{P}$ is a *goal state* if and only if $\mathcal{M} \subseteq S$ and $S \cap \mathcal{N} = \emptyset$.

The effect of a finite sequence of operators $(o_1, o_2, \ldots, o_n)$ on a state S can be formalized as follows:

$$Result(S, ()) = S$$

$$Result(S, (o)) = \begin{cases} (S \cup \alpha) \setminus \delta & \text{if } \varphi \subseteq S \wedge S \cap \eta = \emptyset \\ S & \text{otherwise} \end{cases}$$

$$Result(S, (o_1, o_2, \ldots, o_n)) = Result(Result(S, (o_1)), (o_2, \ldots, o_n))$$

In essence, any operator can be applied to a state, but only has an effect if its preconditions are satisfied. If its preconditions are satisfied, its positive postconditions are added and its negative postconditions are deleted, cf. [Fikes and Nilsson, 1971]. An operator can appear multiple times in a sequence of operators.

A finite sequence of operators $(0_1, 0_2, \ldots, o_n)$ is a *solution* to an instance of prepositional planning if *Result(I,* $(o_1, 0_2, \ldots, o_n)$) is a goal state.

An instance of a propositional planning problem is *satisfiable* if it has a solution. PLANSAT is defined as the decision problem of determining whether an instance of propositional planning is satisfiable. Below, the computational complexity of PLANSAT and restricted versions of PLANSAT are demonstrated.

To show how a planning instance can be modeled by propositional planning, consider the Sussman anomaly. In this blocks-world instance, there are three blocks 4, *B,* and C. Initially *C* is on *A, A* is on the table, and *B* is on the table. The goal is to have A on B, B on C, and *C* on the table. Only one block at a time can be moved. The conditions, initial state, and goals can be represented as follows:

$$\begin{aligned} \mathcal{P} &= \{A\text{-on-}B, A\text{-on-}C, B\text{-on-}A, \\ &\quad B\text{-on-}C, C\text{-on-}A, C\text{-on-}B\} \\ \mathcal{I} &= \{C\text{-on-}A\} \\ \mathcal{M} &= \{A\text{-on-}B, B\text{-on-}C\} \\ \mathcal{N} &= \{A\text{-on-}C, B\text{-on-}A, C\text{-on-}A, C\text{-on-}B\} \end{aligned}$$

The negative goals $\mathcal{N}$ exploit the fact that C is on the table if it is not on top of anything else.

The operator to unstack A can be represented as:

$$\begin{aligned} \varphi &= \emptyset \\ \eta &= \{B\text{-on-}A, C\text{-on-}A\} \\ \alpha &= \emptyset \\ \delta &= \{A\text{-on-}B, A\text{-on-}C\} \end{aligned}$$

That is, A can be moved to the table if nothing is on A. As a result, A will not be on top of any other block.

The operator to stack A on B can be represented as:

$$\begin{aligned} \varphi &= \emptyset \\ \eta &= \{B\text{-on-}A, C\text{-on-}A, A\text{-on-}B, C\text{-on-}B\} \\ \alpha &= \{A\text{-on-}B\} \\ \delta &= \{A\text{-on-}C\} \end{aligned}$$

That is, *A* can be stacked on *B* if nothing is on top of *A* or *B.* The result is that *A* will be on *B* and not on top of any other block.

Obviously, any blocks-world instance can be easily modeled as propositional planning. More generally, any TWEAK planning instance can be polynomially reduced to a propositional planning instance if the initial state is finite, if each variable in an operator schema is limited to a polynomial number of values, and if each operator schema is limited to a constant number of variables. An exponential number of values for a variable would lead to an exponential number of propositional planning conditions. A polynomial number of variables in an operator schema would lead to an exponential number of propositional planning operators.

4 Complexity Results

This section describes and demonstrates our complexity results for propositional planning. Due to space limitations, some of the proofs are overly abbreviated. As mentioned above, PLANSAT is the decision problem of determining whether an instance of propositional planning is satisfiable.

4.1 PSPACE-complete Propositional Planning

Theorem 1 *PLANSAT is PSPACE-complete.*

Proof: PLANSAT is in NPSPACE because a sequence of operators can be nondeterministically chosen, and the size of a state is bounded by the number of conditions. That is, if there are n conditions and there is a solution, then the length of the smallest solution path must be less than 2^n. Any solution of length 2^n or larger must have "loops," i.e., there must be some state that it visits twice. Such loops can be removed, resulting in a solution of length less than 2^n. Hence, no more than 2^n nondeterministic choices are required. Because NPSPACE = PSPACE, PLANSAT is also in PSPACE.

Turing machines whose space is polynomially bounded can be polynomially reduced to PLANSAT. The

PLANSAT conditions can be encoded (and roughly translated) as follows:

$in_{i,x}$	Symbol x is in tape position i.
$at_{i,q}$	The input tape head is at the ith position and the Turing machine is in state q.
$do_{i,q,x}$	Perform the transition at the ith position for state q on character x.
accept	The Turing machine accepts the input.

If q_0 is the initial state of the Turing machine, its input is $x_1 x_2 \ldots x_k$, and the space used by the Turing machine is bounded by n, then the the initial state and goals for propositional planning can be encoded as:

$$\mathcal{I} = \{at_{0,q_0}, in_{0,\#}, in_{1,x_1}, in_{2,x_2}, \ldots, in_{k,x_k}, in_{k+1,\#}, in_{k+2,\#}, \ldots, in_{n-1,\#}\}$$
$$\mathcal{M} = \{accept\}$$
$$\mathcal{N} = \emptyset$$

$\mathcal{I}$ is encoded so that that position 1 to position k contain the input and the remaining positions (position 0 and positions $k+1$ to $n-1$) contain a special symbol #.

Suppose that the Turing machine is in state q, the input tape is at the ith position, x is the character at the ith position, and the transition is to replace x with y, move to the right, and be in state q'. This can be encoded using three operators:

$\varphi = \{at_{i,q}, in_{i,x}\}$ $\eta = \emptyset$ $\alpha = \{do_{i,q,x}\}$ $\delta = \{at_{i,q}\}$

$\varphi = \{do_{i,q,x}, in_{i,x}\}$ $\eta = \emptyset$ $\alpha = \{in_{i,y}\}$ $\delta = \{in_{i,x}\}$

$\varphi = \{do_{i,q,x}, in_{i,y}\}$ $\eta = \emptyset$ $\alpha = \{at_{i+1,q'}\}$ $\delta = \{do_{i,q,x}\}$

The first operator "packs" all the information about the current position into a single condition. The second operator changes the symbol. The third operator moves to the next position and the new state. To handle boundary conditions, encode no operators for $at\ 1_q$ and $at_{n,q}$.

A Turing machine accepts an input if it is in an accepting state and no transition can be made from the current symbol. For each such case, an operator to add *accept* can be encoded,

Because there are a polynomial number of (i,g, x) combinations, there will be a polynomial number of conditions and operators. Thus, any PSPACE Turing machine with its input can be polynomially reduced to a propositional planning instance. □

Note that none of the above operators requires more than two positive preconditions and two postconditions. This leads to the following corollary.

Corollary 2 *PLANS AT with operators restricted to two positive preconditions and two postconditions is PSPACE-complete.*

Using the same conditions as encoded above, the following theorem can be demonstrated:

Theorem 3 *PLANS AT with operators restricted to one postcondition (allowing any number of preconditions) is PSPACE-complete.*

Proof: Let $Do_i = \{do_{u,v,w} \mid u = i\}$. That is, Do_i is the set of all *do* conditions whose first subscript is i. Then the Turing machine transition described above can be encoded using the following six operators:

$\varphi = \{at_{i,q}, in_{i,x}\}$ $\eta = Do_{i-1} \cup Do_{i+1}$ $\alpha = \{do_{i,q,x}\}$ $\delta = \emptyset$

$\varphi = \{at_{i,q}, in_{i,x}, do_{i,q,x}\}$ $\eta = \emptyset$ $\alpha = \emptyset$ $\delta = \{at_{i,q}\}$

$\varphi = \{do_{i,q,x}, in_{i,x}\}$ $\eta = \{at_{i,q}\}$ $\alpha = \{in_{i,y}\}$ $\delta = \emptyset$

$\varphi = \{do_{i,q,x}, in_{i,x}, in_{i,y}\}$ $\eta = \emptyset$ $\alpha = \emptyset$ $\delta = \{in_{i,x}\}$

$\varphi = \{do_{i,q,x}, in_{i,y}\}$ $\eta = \{in_{i,x}\}$ $\alpha = \{at_{i+1,q'}\}$ $\delta = \emptyset$

$\varphi = \{do_{i,q,x}, in_{i,y}, at_{i+1,q'}\}$ $\eta = \emptyset$ $\alpha = \emptyset$ $\delta = \{do_{i,q,x}\}$

In essence, two operators replace each operator in the previous reduction. The structure of the operators ensures that they must be performed in sequence. The key part is the first operator whose negative preconditions include all *do* conditions whose first subscript is *i* — 1 0r i+ 1. This ensures that the *do* condition associated with the previous transition has been removed (see the sixth operator) before the next Turing machine transition hegins. This is why any number of preconditions may be necessary. D

4.2 NP-complete and NP-hard Propositional Planning

Let PLANSAT+ be PLANSAT with operators restricted to positive postconditions.

Theorem 4 *PLANS AT+ is NP-complete.*

Proof: PLANSAT-f- operators can never negate a condition, so a previous state is always a subset of succeeding states. Also, operators within an operator sequence that have no effect can always be removed. Hence, if a solution exists, the length of the smallest solution can be no longer than the number of conditions. Thus, PLANSAT+ is in NP because only a linear number of nondeterministic choices is required.

3SAT can be polynomially reduced to PLANSAT+. 3SAT is the problem of satisfying a formula in prepositional calculus in conjunctive normal form, in which each clause has at most three factors.

Let $\mathcal{F}$ be a formula in propositional calculus in 3SAT form. Let $U = \{u_1, u_2, \ldots, u_m\}$ be the variables usediin $\mathcal{F}$. Let n be the number of clauses in $\mathcal{F}$. An equivalent PLANSAT+ instance can be constructed using the following types of conditions.

T_i	u_i = true is selected.
F_i	u_i = false is selected.
C_j	The jth clause is satisfied.

The initial state and goals can be specified as:

$$\mathcal{I} = \emptyset$$
$$\mathcal{M} = \{C_1, C_2, \ldots, C_n\}$$
$$\mathcal{N} = \emptyset$$

That is, the goal is to satisfy each of the clauses.

For each variable u_i, two operators are needed:

$\varphi = \emptyset$ $\eta = \{F_i\}$ $\alpha = \{T_i\}$ $\delta = \emptyset$

$\varphi = \emptyset$ $\eta = \{T_i\}$ $\alpha = \{F_i\}$ $\delta = \emptyset$

That is, U_i = true can be selected only if u_i - false is not, and vice versa. In this fashion, only one of u_i = true and $U\{$ — false can be selected.

For each case where a clause C_j contains a variable u_i, the first operator below is needed; for a negated variable $\overline{u_i}$, the second operator below is needed:

$$\begin{array}{ll} \varphi = \{T_i\} & \varphi = \{F_i\} \\ \eta = \emptyset & \eta = \emptyset \\ \alpha = \{C_j\} & \alpha = \{C_j\} \\ \delta = \emptyset & \delta = \emptyset \end{array}$$

Clearly, every *Cj* can be made true if and only if a satisfying assignment can be found. Thus PLANSAT+ is NP-hard. Since PLANSAT+ is also in NP, it follows that PLANSAT+ is NP-complete. D

Note that each operator above only requires one precondition and one positive postcondition. This leads to the following corollary.

Corollary 5 *PLANSAT+ with operators restricted to one precondition and one postcondition is NP-complete.*

There is one additional intractability result.

Theorem 6 *PLANSAT with operators restricted to one positive precondition and two postconditions is NP-hard,*

Proof: This can be shown by reduction from 3SAT, similar to that for Theorem 4. One additional type of condition is needed:

U_i The value of U_i is unknown.

The initial state is $\{U_1,..., U_m\}$.

Now all that is needed are different operators for selecting an assignment.

$$\begin{array}{ll} \varphi = \{U_i\} & \varphi = \{U_i\} \\ \eta = \emptyset & \eta = \emptyset \\ \alpha = \{T_i\} & \alpha = \{F_i\} \\ \delta = \{U_i\} & \delta = \{U_i\} \end{array}$$

Again, both u_i = true and U_i — false cannot be selected.

The same operators for clauses as in the proof for Theorem 4 can be used. ❑

4.3 Polynomial Propositional Planning

Theorem 7 *PLANSAT with operators restricted to positive preconditions and one postcondition is polynomial.*

Proof Outline: The difficulty is that some negative goals might need to be temporarily true to make some positive goals true or some negative goals false. Fortunately, because of the restrictions on the operators, it can be shown that any plan can be transformed to another plan that first makes conditions true and then makes conditions false. Thus, if there is a solution, there is a state 5 that meets the following conditions:

S can be reached from the initial state X via operators with positive postconditions;

the positive goals M are a subset of *S;* and

S \ *N,* i.e., a state that satisfies the negative goals, can be reached from 5 via operators with negative postconditions.

Because each operator has only positive preconditions and affects only one condition, and because the positive goals are true of S, the only thing remaining is to make all the negative goals false, i.e., to achieve *S\N.* There is no reason to make other conditions false.

Let *Turnon* then be a subroutine that inputs a set of conditions *X* and returns the maximal state $S \subseteq V \setminus X$ that can be reached from Z. Let *Tumoff* be another subroutine that inputs a set of conditions 5 and returns the maximal state $S' \subseteq S$ such that *S\N* can be reached from *S*,* It can be shown that each subroutine is polynomial, and that each maximal state is unique.

The following algorithm determines if a solution exists by iterating between *Turnon* and *Tumoff:*

$X \leftarrow \emptyset$
loop
  $S \leftarrow Turnon(X)$
  if $\mathcal{M} \not\subseteq S$ **then reject**
  $S' \leftarrow Turnoff(S)$
  if $S = S'$ **then accept**
  $X \leftarrow X \cup (S \setminus S')$
  if $X \cap \mathcal{I} \neq \emptyset$ **then reject**
end loop

It can be shown that every condition added to A' is a condition that, if true, prevents the goal state from being reached. Since *X* grows monotonically, the algorithm is polynomial. O

Theorem 8 *PLANSAT with k goals and operators restricted to one precondition is polynomial.*

Proof: This is the algorithm. Construct all possible combinations of *k* conditions. Mark those combinations true of the Initial state. For each marked combination, mark any combinations that can be reached from that combination via an operator. After all possible combinations are marked, if the combination of conditions corresponding to the goal is marked then accept, otherwise reject.

This is why the algorithm works. Consider a solution plan and any one of the *k* goals. To reach this goal, there must be a "chain" of operators leading from one condition in the initial state through one condition at a time until the goal is reached. Consider now the *k* chains of operators for the *k* goals. Consider also any state reached during the execution of the solution plan. This state will correspond to *k* nodes on the *k* chains. Any state that satisfies the *k* conditions corresponding to those nodes can reach the goal state. Since this is true for all states reached by the solution plan, it must be the case that only *k* conditions at a time need to be considered to determine what combinations of k conditions can be reached. ❑

Note that if operators can have more than one precondition, then a conjunctive goal problem can be converted into a single goal problem by adding operators that map the original set of goals onto a single "supergoal."

Theorem 9 *PLANSAT with operators restricted to no preconditions is polynomial.*

Proof: It is possible to work backwards from the goals. First look for operators that do not clobber any of the goals. Goals that are achieved by these operators can be

removed from consideration. These operators can also be removed from consideration. Then look for operators that do not clobber the remaining goals, and remove from consideration these operators and the goals they achieve. This can be repeated until the remaining goals are true *of* the initial state (accept) or until no more appropriate operators can be found (reject). □

4.4 The Blocks World

Theorem 7 can be used to show that the blocks-world problem is tractable.

Theorem 10 *The blocks-world problem can be solved using operators restricted to positive preconditions and one postcondition.*

Proof: Note that stacking one block on another can be accomplished by first moving the former block on the table and then moving it on top of the latter block. Thus, solving any blocks-world instance only requires operators to move a block to the table and to move a block from the table.

Let $\{B_1, B_2, \ldots\ldots, B_n\}$ be the blocks in an instance of the blocks-world problem. The conditions can be encoded as follows:

$off_{i,j}$ B_i is *not* on top of B_j.

If B_i is on the table, then all $off_{i\,k}$ will be true. If B_i has a clear top, then all off_{ki} will be true. If B_i is on top of B_j, then all $off_{i,k}$ except for $off_{i,j}$ will be true.

For each B_i and B_j, $i \neq j$, the operator to move B_i from on top of Bj to the table can be encoded as:

$$\begin{aligned}\varphi &= \{off_{1,i}, off_{2,i}, \ldots, off_{n,i}, \\ &\quad off_{1,j}, off_{2,j}, \ldots, off_{i-1,j}, off_{i+1,j}, \ldots, off_{n,j}\} \\ \eta &= \emptyset \\ \alpha &= \{off_{i,j}\} \\ \delta &= \emptyset\end{aligned}$$

That is, if nothing is on B_i and nothing is on B_j except possibly B_i, then when this operator is applied, the result is that B_i will not be on top of B_j.

For each B_i and B_j, $i \neq j$, the operator to move B_i from on the table to on top of B_j can be encoded as:

$$\begin{aligned}\varphi &= \{off_{1,i}, off_{2,i}, \ldots, off_{n,i}, \\ &\quad off_{1,j}, off_{2,j}, \ldots, off_{n,j}, \\ &\quad off_{i,1}, off_{i,2}, \ldots, off_{i,n}\} \\ \eta &= \emptyset \\ \alpha &= \emptyset \\ \delta &= \{off_{i,j}\}\end{aligned}$$

That is, if nothing is on Bi and B_j, and if B_i is not on top of any other block, then when this operator is applied, the result is that B_i will be on top of Bj.

Since there are only $0(n^2)$ (i,j) combinations, oniy $0(n^2)$ conditions and operators are needed to encode a blocks-world instance.

As required, all preconditions are positive and each operator has only one postcondition. Thus, Theorem 7, in a sense, explains why the blocks world is tractable.[3]

[3]The algorithm for Theorem 7 corresponds to the unimaginative, but robust, strategy of moving all the blocks to the table, which makes all the conditions positive, and then forming the stacks from the table on up.

5 Remarks

Planning is intractable even if the size of states are bounded and operators have no variables. Merely allowing two preconditions and two postconditions for operators gives rise to an extremely hard problem, However, operators must have preconditions, postconditions, and apparently many more "features'[1] to implement any interesting domain [Chapman, 1987; Hendler *et al.,* 1990], While additional features might be good for making a planner more useful as a programming tool, generality has its downside—tractability cannot be guaranteed unless there are sufficient restrictions on the operators.

Some restricted propositional planning problems are tractable. Restricting operators to positive preconditions and one postcondition explains the tractability of the blocks-world. Restricting operators to one precondition and limiting the number of goals is the only interesting case where restricting the number of goals leads to tractability. Restricting operators to no preconditions shows that planning is tractable if preconditions can be ignored, e.g., if preconditions of operators can be easily satisfied without clobbering already achieved goals.

However, many, if not most, planning domains violate these categories. Thus, these domains cannot be shown to be tractable based on restrictions on the local properties of operators. As mentioned in the section on previous research, Korf [1987] lists several global properties of planning problems that lead to efficient search for plans. Understanding how these properties are realized as restrictions on the set of operators as a whole is a promising research approach.

Unfortunately, the prospects for a single domain-independent planning algorithm are pessimistic. The three tractable problems above appear to require quite different algorithms, and many other tractable planning problems are yet to be discovered. This indicates that it will be more fruitful to adopt different algorithms for different types of planning problems.

References

[Allen *et* al., 1990] Allen, J.; Hendler, J.; and Tate, A., editors 1990. *Readings in Planning,* Morgan Kaufmann, San Mateo, CA.

[Chapman, 1987] Chapman, D. 1987. Planning for conjunctive goals. *Artif. Intell.* 32(3):333-377.

Dean and Boddy, 1987] Dean, T. and Boddy, M, 1987. Reasoning about partially ordered events. *Artif. Intell* 36(3):375-399.

Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving *to* problem solving, *Artif. Intell.* 2(3/4):189-208-

[Hendler *et* al., 1990] Hendler, J.; Tate, A.; and Drummond, M. 1990. AI planning: Systems and techniques. *AI Magazine* U(2):6K77.

[Korf, 1985] Korf, R. E. 1985. Macro-operators: A weak method for learning. *Artif. Intell.* 26(I):35-77.

Korf, 1987] Korf, R. E. 1987. Planning as search: A quantitative approach. *Artif. Intell.* 33(I):65-88.