The Computational Complexity of Propositional STRIPS Planning

Tom Bylander
Division of Mathematics, Computer Science, and Statistics
University of Texas at San Antonio
San Antonio, Texas 78249
USA
email: bylander@ringer.cs.utsa.edu
phone: 210-691-5693

March 7, 1994

to appear in *Artificial Intelligence*

Abstract

I present several computational complexity results for propositional STRIPS planning, i.e., STRIPS planning restricted to ground formulas. Different planning problems can be defined by restricting the type of formulas, placing limits on the number of pre- and postconditions, by restricting negation in pre- and postconditions, and by requiring optimal plans. For these types of restrictions, I show when planning is tractable (polynomial) and intractable (*NP*-hard). In general, it is *PSPACE*-complete to determine if a given planning instance has any solutions. Extremely severe restrictions on both the operators and the formulas are required to guarantee polynomial time or even *NP*-completeness. For example, when only ground literals are permitted, determining plan existence is *PSPACE*-complete even if operators are limited to two preconditions and two postconditions. When definite Horn ground formulas are permitted, determining plan existence is *PSPACE*-complete even if operators are limited to zero preconditions and one postcondition. One of the interesting tractable problems is if each operator is restricted to positive preconditions and one postcondition (only ground literals). The blocks-world problem, slightly modified, is a subproblem of this restricted planning problem. These results in combination with previous analyses are not encouraging for domain-independent planning.

*This research has been supported in part by DARPA/AFOSR contract F49620-89-C-0110.

†This paper is a revised, integrated, and extended version of [5] and [6].

1 Précis

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, when partial or approximate solutions are acceptable, heuristic theories are a valid approach, i.e., theories for efficiently solving most, but not all, instances. 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, both optimal and non-optimal, in which all instances can be efficiently solved.

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 [9]. However, there are few results that provide clear dividing lines between tractable and intractable planning. Below, I clarify several of these dividing lines by analyzing the computational complexity of a planning problem and a variety of restricted and augmented versions. From the perspective of the expressiveness-tractability tradeoff [27], I primarily consider the expressiveness of operators and formulas.

1.1 Previous Research

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. [1] and Hendler et al. [23]. Despite the sizable literature, results on computational complexity are sparse.

Dean and Boddy [12] 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). 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 [25] 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, I focus exclusively on local properties of operators. However, except for Korf's own analysis of operator decomposability [24], neither he nor I describe the relationship from these properties of planning problems to the properties of operators. Clearly, this is an issue that future work should address.

Perhaps the most important complexity results for planning are due to Chapman's analysis of TWEAK [9]. Because virtually all other planners are as expressive as TWEAK, Chapman's 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 did not show what features of TWEAK's representation are to blame for the complexity. Erol et al. [16] analyze TWEAK-like planning further, showing that planning is undecidable only if the number of constants is infinite. They also demonstrate that if the set of constants are finite, the operators are fixed, no functions are allowed, and no negative pre- or postconditions are permitted, then TWEAK-like planning is polynomial. However, this is a very specialized kind of planning problem, and it is not clear how some of these conditions can be relaxed.

There are also some results concerning the tractability of very specialized kinds of planning. In the case where states are value assignments to finite-valued state variables, Bäckström and Klein [3] show that planning is tractable if each operator has one postcondition, i.e., changes the value of one variable, if the preconditions of any two operators do not require different values for non-changing variables, and if no two operators have the same postcondition. Ratner and Warmuth [34] show that finding optimal solutions to the $n \times n$ generalization of the 8-puzzle is *NP*-hard. Gupta and Nau [21] and Chenoweth [11] show that optimal blocks-world planning is *NP*-hard. Bacchus and Yang [2] present tractable tests for determining when an hierarchical planning problem has a property called downward refinement, i.e., every abstract solution can be refined into a concrete solution. While these results provide valuable insight into specialized problems, they provide little information about the complexity of planning as a whole.

1.2 Models of Planning

I analyze two closely related models of planning. Both models are impoverished compared to working planners. They are intended to be tools for theoretical analysis rather than programming convenience. The results for these models apply to first-order STRIPS planning [28] when there is a limited number of relevant ground formulas.

The first model of planning, called "propositional STRIPS planning," is STRIPS planning [17] in which an initial state is a finite set of ground atomic formulas, indicating that the corresponding conditions are initially true, and that all other relevant conditions are initially false; the preconditions and postconditions of an operator are ground literals; and the goals are ground literals. Operators in this model do not have any variables or indirect side effects. First-order STRIPS planning can be reduced to propositional STRIPS planning if the initial state and goal conditions are ground literals, all pre- and postconditions of operators are literals, and the number of relevant ground atomic formulas is limited. See Section 2 for a more complete description.

The second model of planning, called "extended propositional STRIPS planning," augments propositional STRIPS planning with a "domain theory" for inferring additional effects, where a domain theory is a set of ground formulas. The ramification problem[1] is finessed by requiring a preference ordering of all the literals so that, roughly, if two literals are true of

[1] The ramification problem is that the effects of an operator in the context of a formula can be ambiguous [19, 20]. For example, if $A \vee B \vee C$ is a formula in the domain theory, and if an operator deletes A, it is ambiguous whether B or C should result.

the previous state, and if it is inconsistent to assert both in the next state, then the ordering specifies which literal remains true. The preference ordering ensures that the result of applying an operator is unambiguous. First-order STRIPS planning can be reduced to extended propositional STRIPS planning if the number of relevant ground formulas is limited.[2] See Section 5 for a more complete description.

Propositional STRIPS planning is equivalent to Nilsson's [33] simplified description of STRIPS except that propositional STRIPS planning requires that each planning instance explicitly specifies all relevant ground atomic formulas. Extended propositional STRIPS planning is most closely related to Ginsberg and Smith's [20] possible world approach to reasoning about actions. Each state in extended propositional STRIPS planning corresponds to a "partial world" with the domain theory as "protected" formulas, and the state resulting from operator application corresponds to a "possible world." Besides the fact that they use first-order formulas, the other major difference is that the resulting partial world for Ginsberg and Smith is taken to be the agreement among "all possible worlds," which leads to ambiguity where the possible worlds conflict, rather than preferring a particular possible world.

1.3 Summary of Results

Different planning problems can be defined by placing limits on the number of pre- and postconditions, by restricting negation in pre- and postconditions, by requiring optimal plans, and by restricting the domain theory. Figures 1, 2, and 3 summarize the results. A few of the results are due to Erol et al. [16], and are footnoted accordingly.

1.3.1 PLANSAT

Let PLANSAT be the problem of determining the existence of a solution for propositional STRIPS planning. Figure 1 illustrates the complexity of PLANSAT under various restrictions, showing which PLANSAT problems are *PSPACE*-complete, *NP*-complete, and polynomial.[3] Section 3 contains the proofs for these results.

Each box in the figure denotes limitations on the number of pre- and postconditions—a * indicates no limits. Also, some boxes denote restrictions on negation using the + symbol. For example, the following box:

2 + preconds 2 postconds.

[2] This assertion requires at least two qualifications. One is that I assume that all non-literal formulas in the initial state and in the postconditions of operators are true in all states, as is done in Lifschitz's formalization of the semantics of STRIPS planning [28]. The other is that I assume (speculatively) that any solution to the ramification problem can be implemented in any given domain as a preference ordering of literals plus appropriate formulas in the domain theory.

[3] A problem is in *PSPACE* if it can be solved using an amount of space that is a polynomial of the size of the input. *PSPACE*-complete problems are the hardest problems in *PSPACE*. As is customary, I assume that *PSPACE*-complete problems are harder than *NP*-complete problems, which in turn are harder than polynomial problems. However, even P $\neq$ *PSPACE* is not yet proven.

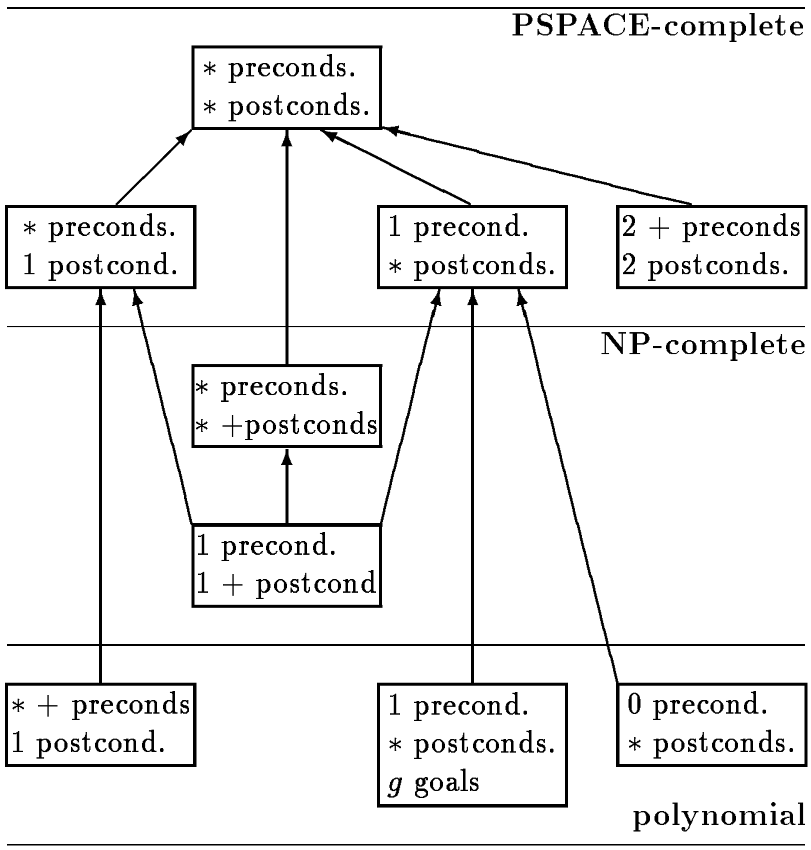

Figure 1: Complexity Results for PLANSAT

says that operators are limited to two positive preconditions and two postconditions. One box in Figure 1 indicates that the number of goals is bounded by a constant g. The arrows indicate subproblem relationships.[4]

PLANSAT is in *PSPACE* because the size of a state is bounded by the number of conditions. Although the length N of a solution plan might be exponential, an algorithm only needs memory for $O(\lg N)$ states to determine if a solution exists [35]. Namely, an algorithm PLAN-EXISTS(S_1, S_2, N) that determines the existence of a plan from state S_1 to state S_2 of length N or less can be implemented as follows: for $N = 1$, check if $S_1 = S_2$ or if applying any operator to S_1 results in S_2; for $N > 1$ iterate over all states S_3, and recursively call PLAN-EXISTS$(S_1, S_3, \lceil N/2 \rceil)$ and PLAN-EXISTS$(S_3, S_2, \lfloor N/2 \rfloor)$). Note that the depth of the recursion is logarithmic in N.

[4]The following are other results that were left out the figure because they were judged to be less interesting, but are listed here for completeness. Their proofs are omitted from the paper. For any constant $k \geq 1$, PLANSAT is *NP*-hard if each operator is limited to k preconditions and one postcondition. For any constant $k \geq 1$, PLANSAT is *NP*-hard if each operator is limited to one precondition and k postconditions. PLANSAT is *NP*-complete if each operator is limited to positive preconditions and negative postconditions (likewise negative preconditions and positive postconditions), even if limited to one positive precondition and two negative postconditions. PLANSAT is polynomial in the previous case if the number of goals is bounded by a constant. PLANSAT is polynomial if each operator is limited to positive preconditions and positive postconditions. Erol et al. [16] show that PLANSAT is *NLOGSPACE*-complete if each operator is limited to one positive precondition and positive postconditions.

PLANSAT is *PSPACE*-hard because any Turing machine transition from one state to another can be mapped into an operator. The number of such operators is proportional to the number of transitions times the number of tape squares, so a *PSPACE* Turing machine corresponds to a polynomial number of operators.

PLANSAT remains *PSPACE*-complete even if operators are limited to one postcondition, or if operators are limited to one precondition,[5] or if operators are limited to 2 positive preconditions and 2 postconditions. Section 3 provides a Turing machine reduction for each restriction.

A necessary condition to make PLANSAT *PSPACE*-complete is allowing negative postconditions. If operators are limited to positive postconditions, then applying an operator to a state must result in a superset of that state. This means that the length of the shortest solution plan (if one exists) is bounded by the number of conditions (ground atomic formulas). Thus, PLANSAT limited to positive postconditions is in *NP*. Note that if no two operators of a planning instance have conflicting postconditions (a postcondition "conflict" is when a postcondition of one operator is the negation of a postcondition of another operator), then it is easy to transform the instance so that all postconditions are positive.

However, this restriction is still *NP*-complete because a positive postcondition can conflict with negative preconditions of other operators, and it can be difficult to choose which conditions should be made true in order to achieve the goals.

Thus, further restrictions are required to guarantee polynomial planning. The first polynomial problem (bottom left of Figure 1) is if each operator only has positive preconditions and a single postcondition. For this problem, an algorithm can search for an intermediate state that can be reached from the initial state via operators with positive postconditions and that can reach a goal state via operators with negative postconditions. The blocks-world problem, slightly modified, is a subproblem of this problem. Note that one can always unstack all the blocks followed by making the goal stacks from the table on up. The intermediate state in this case is when all the blocks are unstacked. Bäckström and Nebel [4] give a generalization of this restriction and the corresponding algorithm in which state variables are used instead of propositions.

The second polynomial problem is if there are a limited number of goals and each operator is limited to one precondition. Surprisingly, limiting the number of goals is an important factor only if operators are limited to one precondition. Otherwise, any single goal can expand into many subgoals. This problem is polynomial because a polynomial-sized search space can be constructed, i.e., g goals depend on at most g conditions for any state, so an algorithm need only consider all combinations of g conditions.

The final polynomial problem is if operators have no preconditions at all, i.e., planning is tractable if preconditions can be ignored. Interestingly, this is the only polynomial problem that appears to be easily solvable by means-end analysis. The first polynomial problem appears to require significant forward search. The second appears to require an exhaustive search through a reduced search space.

Not illustrated in Figure 1 is that all of these results are independent of whether the planner is linear, non-linear, deliberative, reactive, anytime, hierarchical, opportunistic, case-based, etc. It does not even matter whether the operators are explicitly represented or

[5]In previous papers, I incorrectly asserted that the complexity of this restriction was *NP*-complete.

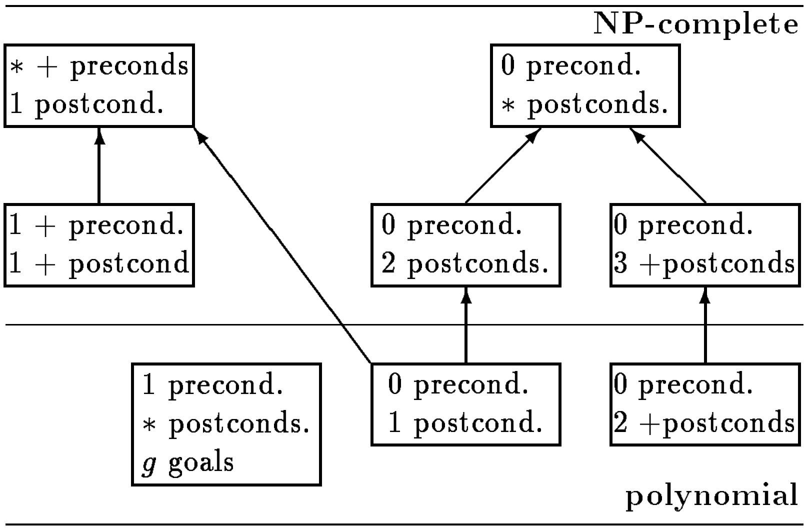

Figure 2: Complexity Results for PLANMIN

implicitly available via some result function. Thus, even in the simplest case, i.e., a restriction to ground literals, full knowledge of initial state and operators, deterministic operators, and only asking whether any solution exists, planning is a very hard problem.

1.3.2 PLANMIN

To describe the computational complexity of an optimization problem, it is converted into the decision problem of determining whether a given bound can be achieved [18]. So to describe the complexity of optimal planning, let PLANMIN be the problem of determining the existence of a solution of k operators or less for propositional STRIPS planning, where k is given as part of the input. The proofs for these results are in Section 4.

For each PLANSAT problem that was *PSPACE*(*NP*)-complete, it is also *PSPACE*(*NP*)-complete for PLANMIN.[6] This is because one can easily can set k high enough so that a PLANMIN instance is equivalent to the corresponding PLANSAT instance. Also, the *NP*-complete problems have polynomial bounds on the length of the shortest solution plan, if such a plan exists. Figure 2 illustrates the complexity results for the restrictions that were polynomial for PLANSAT.

When operators are restricted to one positive precondition and one positive postcondition, PLANMIN remains intractable. The difficulty is finding a minimum set of subgoals to achieve a set of goals. That is, the postcondition of one operator might be useful to achieve several goals via other operators, and it is difficult to choose a minimum set of such conditions. This improves on the result in Erol et al. [16], who show that PLANMIN is *NP*-complete if each operator is limited to one positive precondition and positive postconditions.

[6]Erol et al. [16] also analyze the complexity of PLANMIN (which they call PLAN LENGTH), but, with one exception described in the next paragraph, they only consider restrictions on negation in preconditions and postconditions. Based on my results for PLANSAT [5], they show that PLANMIN is *PSPACE*-complete, that it is *PSPACE*-complete if each operator is limited to positive preconditions, and that it is *NP*-complete if each operator is limited to positive postconditions. Erol et al. also show that PLANMIN is *NP*-complete if each operator is limited to positive preconditions and positive postconditions.

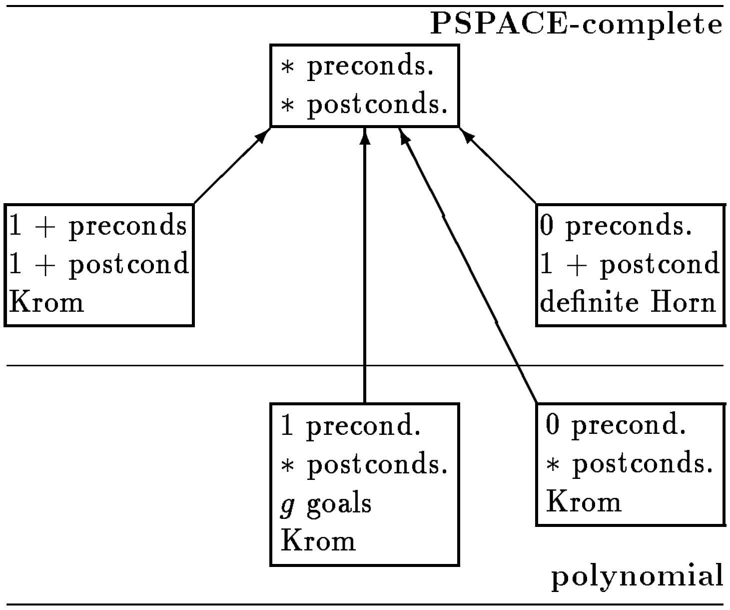

Figure 3: Complexity Results for EPLANSAT

Even when operators are restricted to no preconditions, restrictions on postconditions are required for polynomial optimal planning. If two postconditions or three positive postconditions are allowed, then PLANMIN is *NP*-complete.

The only interesting polynomial PLANMIN problem is limiting the number of goals and limiting operators to one precondition. This is because the relevant search space has polynomial size and breadth-first search can be used to find the shortest path.

1.3.3 EPLANSAT

Both PLANSAT and PLANMIN are restricted to ground literals. If ground formulas are allowed, then extended propositional STRIPS planning must be considered. Let EPLANSAT be the problem of determining the existence of a solution for extended propositional STRIPS planning. Figure 3 illustrates the results. "definite Horn" denotes a restriction to propositional definite-Horn domain-theories (every formula is a disjunction of ground literals with exactly one positive literal); "Krom" denotes a restriction to propositional Krom domain theories (every formula is a disjunction of at most two ground literals).[7] These results are proven in Section 6.

As shown by the figure, the range of *PSPACE*-complete problems is large, from EPLANSAT without restrictions to EPLANSAT limited to definite Horn theories and operators with zero preconditions and one positive postcondition. Essentially, definite Horn theories combined with one condition changing at a time is sufficient to permit the same effects as PLANSAT operators. Applying an operator limited to one positive postcondition does not necessarily result in a state that is superset of the previous state. This is because conditions can be made false by an interaction of the domain theory with the preference ordering of literals.

For a Krom domain theory, a postcondition always implies the same set of conditions

[7]The use of "Krom" to refer to this type of formula can be traced to a paper by Krom [26].

regardless of the current state and other postconditions. Thus, any EPLANSAT problem restricted to Krom theories is equivalent to the corresponding PLANSAT problem without any limitations on the number or type of postconditions. All the Krom theory results are then easily derived from PLANSAT results (see Figure 1): having one precondition is *PSPACE*-complete, one precondition with a bounded number of goals is polynomial, and having no preconditions is polynomial.

1.3.4 EPLANMIN

Let EPLANMIN be the problem of determining the existence of a solution of k operators or less for extended propositional STRIPS planning. For each *PSPACE*(*NP*)-complete EPLANSAT problem, it is also *PSPACE*(*NP*)-complete for EPLANMIN. The two problems that are polynomial for EPLANSAT have different results for EPLANMIN:

> With Krom theories, EPLANMIN is *NP*-complete if operators are limited to no preconditions, even if further limited to one positive postcondition.
>
> With Krom theories, EPLANMIN is polynomial if operators are limited to 1 precondition and the number of goals is bounded by a constant g.

Section 7 demonstrates these results.

1.4 Remarks

I have shown that propositional STRIPS planning is *PSPACE*-complete in general and for many restrictive problems. Extremely severe restrictions on both the operators and the domain theory are required to guarantee polynomial time or even *NP*-completeness. To say the least, these results in combination with previous analyses are not encouraging for domain-independent planning.

However, operators must have multiple preconditions, simultaneous positive and negative postconditions, and apparently many more "features" to implement any interesting domain [9, 23]. While additional features might be good for making a planner more useful as a programming tool, generality has its downside—tractability of planning cannot be guaranteed even with moderately expressive operators.

All of this again calls into question the "restricted language thesis" proposed by Levesque and Brachman [27], i.e., that the expressiveness of representations needs to be restricted so that inference is tractable. Doyle and Patil [14] persuasively argue that restricting the expressiveness of a general-purpose representation system to ensure polynomial reasoning "destroys the generality of the language and the system" and "fails to permit expression of concepts necessary to some applications." The above complexity results for planning provide additional evidence against the restricted language thesis. Restricting the expressiveness of a general-purpose planner to ensure polynomial planning no doubt would destroy the generality of the planner and would fail to permit expression of crucial domain knowledge.

However, simply throwing out the restricted language thesis is no solution. Understanding how humans and robots can reason efficiently will continue to remain a central problem of AI. If restricting the expressiveness of planners won't work, what will?

One possible solution is to consider average running time rather than worst-case running time, so that more expressive languages would be labeled efficient. However, while it appears that *NP*-complete problems are hard only for narrow ranges of the problems [10, 30, 31], there is little research on problems that are *PSPACE*-complete. Promising directions include Musick and Russell [32], who develop a Markov model approximation for analyzing hill-climbing algorithms on single postcondition problems, and Bylander [8], who shows that most PLANSAT instances under certain distributional assumptions can be efficiently solved.

An alternative approach is to restrict global properties of planning instances rather than local properties of operators and formulas. As mentioned above, Korf [25] discusses how global properties such as serializable subgoals, operator decomposability, and abstraction can 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. Korf's [24] analysis of serial decomposability and Bacchus and Yang's [2] analysis of abstraction are significant steps in this direction, though see [7] for negative results on the complexity of serial decomposability.

More generally, I speculate that the analysis of general-purpose planning will evolve into the analysis of many different special-purpose planning problems and techniques for encoding these special-purpose problems in a general-purpose planner. That is, the situation will be like that of analysis of algorithms, in which a general-purpose programming language is used to encode the algorithms and data structures for different types of problems. For example, the three polynomial PLANSAT problems appear to require quite different algorithms and data structures, and many other tractable planning problems are yet to be discovered.

This paper provides a complexity "map" of the "territory" of planning problems, identifying many of the conditions that differentiate tractable planning from intractable planning. Any map, of course, is an incomplete description; in this case, I have focused on restricting the local properties of planning operators and of the formulas in domain theories. Nevertheless, travelers into the planning "wilderness" would be well-advised to be aware of the known computational cliffs.

1.5 A Guide to the Rest of the Paper

All of the above is intended to be an adequate description of the complexity results for readers who do not wish to delve into the formal details. What follows provides the mathematical definitions and the proofs demonstrating the results. Two of the sections below are devoted to defining propositional STRIPS planning and extended propositional STRIPS planning. Each of these two sections also has a blocks-world example and describes the scope of the planning model. All other sections provide proofs for the variety of PLANSAT, PLANMIN, EPLANSAT, and EPLANMIN problems described above. The proofs for the polynomial planning problems include descriptions of algorithms that are sufficient for the proofs, but not necessary the most efficient in terms of running time or plan length.

2 Propositional STRIPS Planning

In this section, I define propositional STRIPS planning, give an example, and describe the kinds of first-order STRIPS planning that can be reduced to propositional planning.

2.1 Definitions

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

> $\mathcal{P}$ is a finite set of ground atomic formulas, called the *conditions*;
>
> $\mathcal{O}$ is a finite set of *operators*, where each operator o has the form $Pre \Rightarrow Post$:[8]
>
> > Pre is a satisfiable conjunction of positive and negative conditions, respectively called the *positive preconditions* (o^+) and the *negative preconditions* (o^-) of the operator;
> >
> > $Post$ is a satisfiable conjunction of positive and negative conditions, respectively called the *positive postconditions* (o_+) and the *negative postconditions* (o_-) of the operator, i.e., the add list and delete list, respectively;
>
> $\mathcal{I} \subseteq \mathcal{P}$ is the *initial state*; and
>
> $\mathcal{G}$, called the *goals*, is a satisfiable conjunction of positive and negative conditions, respectively called the *positive goals* ($\mathcal{G}_+$) and the *negative goals* ($\mathcal{G}_-$).

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 $p \in S$, and false otherwise. $\mathcal{O}$ 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 S satisfies $\mathcal{G}$, i.e., if $\mathcal{G}_+ \subseteq S$ and $\mathcal{G}_- \cap S = \emptyset$.

As indicated above, let o^+, o^-, o_+, and o_- respectively denote the sets of positive preconditions, negative preconditions, positive postconditions, and negative postconditions of an operator o. Then the effect of a finite sequence of operators $(o_1, o_2, \ldots, o_n)$ on a state S can be formalized as follows.[9]

$$
\begin{aligned}
&\mathit{Result}(S, (\,)) = S \\
&\mathit{Result}(S, (o)) = \begin{cases} (S \cup o_+) \setminus o_- & \text{if } o^+ \subseteq S \text{ and } o^- \cap S = \emptyset \\ S & \text{otherwise} \end{cases} \\
&\mathit{Result}(S, (o_1, o_2, \ldots, o_n)) \\
&\qquad = \mathit{Result}(\mathit{Result}(S, (o_1)), (o_2, \ldots, o_n))
\end{aligned}
$$

For convenience, 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

[8] The symbol $\Rightarrow$ is used for operators. The symbol $\rightarrow$ is used for implication.

[9] The symbol $\setminus$ is used for set difference.

added and its negative postconditions are deleted. An operator can appear multiple times in a sequence of operators.

A finite sequence of operators $(o_1, o_2, \ldots, o_n)$ is a *solution* to an instance of propositional planning if $\mathit{Result}(\mathcal{I}, (o_1, o_2, \ldots, o_n))$ is a goal state.

An instance of a propositional STRIPS planning problem is *satisfiable* if it has a solution. PLANSAT is defined as the decision problem of determining whether an instance of propositional STRIPS planning is satisfiable. PLANMIN is defined as the decision problem of determining whether an instance of propositional STRIPS planning has a solution of length k or less, where k is given as part of the input. Below, the computational complexity of PLANSAT, PLANMIN, and restricted versions are demonstrated.

2.2 Blocks-World Example

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

$$\begin{aligned}
\mathcal{P} &= \{\mathit{on}(A,B),\ \mathit{on}(A,C),\ \mathit{on}(B,A),\ \mathit{on}(B,C),\ \mathit{on}(C,A),\ \mathit{on}(C,B),\\
&\qquad \mathit{clear}(A),\ \mathit{clear}(B),\ \mathit{clear}(C),\ \mathit{on}(A,\mathit{Table}),\ \mathit{on}(B,\mathit{Table}),\ \mathit{on}(C,\mathit{Table})\}\\
\mathcal{I} &= \{\mathit{clear}(C),\ \mathit{on}(C,A),\ \mathit{on}(A,\mathit{Table}), \mathit{clear}(B),\ \mathit{on}(B,\mathit{Table})\}\\
\mathcal{G} &= \mathit{on}(A,B) \wedge \mathit{on}(B,C)
\end{aligned}$$

The operator to move C from A to the table can be represented as:

$$\mathit{clear}(C) \wedge \mathit{on}(C,A) \Rightarrow \mathit{on}(C,\mathit{Table}) \wedge \mathit{clear}(A) \wedge \overline{\mathit{on}(C,A)}$$

That is, C can be moved from A to the table if C is clear and C is on A. As a result, C will be on the table, A will be clear, and C will not be on A.

The operator to move A from the table to B can be represented as:

$$\mathit{clear}(A) \wedge \mathit{on}(A,\mathit{Table}) \wedge \mathit{clear}(B) \Rightarrow \mathit{on}(A,B) \wedge \overline{\mathit{on}(A,\mathit{Table})} \wedge \overline{\mathit{clear}(B)}$$

That is, A can be moved from the table to B if nothing is on A or B, and A is on the table. The result is that A will be on B, A will not be on the table, and B will not be clear.

2.3 Reducibility

Any blocks-world instance can be reduced in polynomial time to an instance of propositional STRIPS planning. For b blocks, the above style of encoding will lead to $b^2 + b$ conditions and $b^3 - b^2$ operators. Specifically, there are b *clear* conditions and b^2 *on* conditions (choose any block for the first argument; choose any other block or the table for the second argument). There are $b(b-1)$ operators for moving a block from another block to the table, $b(b-1)$ operators for moving a block from the table to another block, and $b(b-1)(b-2)$ operators for moving a block from one block to another block.

More generally, first-order STRIPS planning can be polynomially reduced to propositional STRIPS planning under the following restrictions: the initial state and goal conditions are ground literals, pre- and postconditions in operators are limited to literals; each variable in an operator is limited to a polynomial number of values; and each operator is limited to a constant number of variables [16]. An exponential number of values for a variable would lead to an exponential number of ground atomic formulas. A polynomial number of variables in an operator would lead to an exponential number of propositional STRIPS planning operators.

Thus, results for propositional STRIPS planning apply to a large range of first-order STRIPS planning, though it should be noted that no formulas are permitted. Extended propositional STRIPS planning shall partially alleviate this restriction.

3 Complexity Results for PLANSAT

This section describes and demonstrates complexity results for PLANSAT, the decision problem of determining whether an instance of propositional STRIPS planning is satisfiable. First, unrestricted PLANSAT is considered, followed by increasingly restrictive versions. The notation PLANSAT$_\beta^\alpha$ is used to denote PLANSAT with restrictions α on the preconditions and restrictions β on the postconditions. For example, PLANSAT$_2^{2+}$ denotes PLANSAT with operators limited to two positive preconditions and two postconditions.

3.1 PSPACE-complete PLANSAT

Theorem 3.1 *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 $NPSPACE = PSPACE$ [35], 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 Turing machine is ready to perform the transition for the current position i and 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 $x_1 x_2 \ldots x_n$, and the space used by the Turing machine is bounded by m, then the the initial state and goals for propositional planning can be encoded as:

$$\begin{aligned}\mathcal{I} = \{&at(1, q_0),\ in(0, \#),\ in(1, x_1),\ in(2, x_2),\ \ldots,\ in(n, x_n),\\ &in(n+1, \#),\ in(n+2, \#),\ \ldots,\ in(m-1, \#)\}\\ \mathcal{G} = {}&accept\end{aligned}$$

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

Suppose that the Turing machine is in state q, the tape head 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 (in order to facilitate corollaries and theorems to follow):

$$at(i,q) \wedge in(i,x) \Rightarrow do(i,q,x) \wedge \overline{at(i,q)}$$

$$do(i,q,x) \wedge in(i,x) \Rightarrow in(i,y) \wedge \overline{in(i,x)}$$

$$do(i,q,x) \wedge in(i,y) \Rightarrow at(i+1,q') \wedge \overline{do(i,q,x)}$$

The first operator "packs" all the information about the current position ($at(i,q)$ and $in(i,x)$) into a single condition $do(i,q,x)$ and deletes $at(i,q)$. The second operator changes the symbol from x to y. The third operator moves to the next position and the new state. To handle boundary conditions, encode no operators for $at(-1,q)$ and $at(m,q)$.

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 operators can precisely encode the transitions and the detection of accepting states, a satisfiable plan can be found if and only if the Turing machine halts in an accepting state. Because there are a polynomial number of (i,q,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 an instance of PLANSAT. □

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

Corollary 3.2 *$PLANSAT_2^{2+}$ is* PSPACE*-complete.*

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

Theorem 3.3 *$PLANSAT_1$ (operators limited 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 argument is i. Then the Turing machine transition described above can be encoded using the following six operators:

$$at(i,q) \wedge in(i,x) \wedge \bigwedge_{p \in Do_{i-1}} \overline{p} \ \wedge \bigwedge_{p \in Do_{i+1}} \overline{p} \Rightarrow do(i,q,x)$$

$$at(i,q) \wedge in(i,x) \wedge do(i,q,x) \Rightarrow \overline{at(i,q)}$$

$$do(i,q,x) \wedge in(i,x) \wedge \overline{at(i,q)} \Rightarrow in(i,y)$$

$$do(i,q,x) \wedge in(i,x) \wedge in(i,y) \Rightarrow \overline{in(i,x)}$$

$$do(i,q,x) \wedge in(i,y) \wedge \overline{in(i,x)} \Rightarrow at(i+1,q')$$

$$do(i,q,x) \wedge in(i,y) \wedge at(i+1,q') \Rightarrow \overline{do(i,q,x)}$$

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 $i-1$ or $i+1$. This ensures that the *do* condition associated with the previous transition has been removed (see the sixth operator) before the next Turing machine transition begins. This is why any number of preconditions is necessary (for this reduction). □

Again using the same encoding as in the proof of Theorem 3.1, the following can be demonstrated.

Theorem 3.4 *PLANSAT*^{1} *is* PSPACE-*complete.*

Proof: For this reduction, each Turing machine must be modified so that if it accepts the input, then the tape is left blank (e.g., filled in with the special symbol #), the Turing machine is in a special state q_f, and the tape head is at position 1. This type of modification does not require significantly more space. If the space used by the Turing machine is bounded by m, then the corresponding goal state for the planning instance is:

$$\mathcal{G} = at(1,q_f) \wedge in(0,\#) \wedge in(1,\#) \wedge \ldots \wedge in(m-1,\#)$$

Let $Do_{i,q,x} = \{do(u,v,w) \mid u \neq i \wedge v \neq q \wedge w \neq x\}$ (all *do* conditions except for $do(i,q,x)$). Let $At = \{at(u,v)\}$ (all *at* conditions). Let $In_{i,x} = \{in(u,v) \mid u = i \wedge v \neq x\}$ (all *in* conditions whose first argument is i and second argument is not x).

Now the Turing machine transition described above can be encoded using the following three operators:

$$at(i,q) \Rightarrow do(i,q,x) \wedge \overline{at(i,q)} \wedge \bigwedge_{p \in In_{i,x}} \overline{p}$$

$$in(i,x) \Rightarrow in(i,y) \wedge \overline{in(i,x)} \wedge \bigwedge_{p \in At} \overline{p} \wedge \bigwedge_{p \in Do_{i,q,x}} \overline{p}$$

$$do(i,q,x) \Rightarrow at(i+1,q') \wedge \overline{do(i,q,x)} \wedge \bigwedge_{p \in In_{i,y}} \overline{p}$$

These three operators correspond to the three operators in the reduction for Theorem 3.1. Note that each operator adds at most one condition and deletes at least one condition. Because the goal state requires as many positive conditions as in the initial state, each operator must add one condition and delete only one condition. In effect, $in(i,x)$ must be true to apply the first operator (otherwise, no *in* condition would be true for the ith position). $do(i,q,x)$ must be true to apply the second operator (otherwise, no *at* or *do* condition would be true). Finally, $in(i,y)$ must be true to apply the third operator (otherwise, no *in* condition would be true for the ith position). Consequently, these three operators must be used as if they were the three operators from the proof for Theorem 3.1. □

Note that the second operator includes all but one of the *at* and *do* conditions. This is why any number of postconditions is necessary (for this reduction). Also, note that there are multiple goals. As shown below, if there is a limit on the number of goals, the problem becomes polynomial.

I have not determined the precise complexity of PLANSAT_1^k and PLANSAT_k^1 for when k is a constant, $k \geq 1$. I speculate that these problems fall into the polynomial hierarchy in a regular way, but the results in this paper only show that they are *NP*-hard and in *PSPACE*.

3.2 NP-Complete PLANSAT

Theorem 3.5 *$PLANSAT_+$ is* NP*-complete.*

Proof: Operators without negative postconditions 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 propositional calculus in conjunctive normal form, in which each clause has at most three literals.

Let $\mathcal{F}$ be a formula in propositional calculus in 3SAT form. Let $U = \{u_1, u_2, \ldots, u_m\}$ be the variables used in $\mathcal{F}$. 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:

$$\mathcal{I} = \emptyset$$
$$\mathcal{G} = C(1) \wedge C(2) \wedge \ldots \wedge C(n)$$

That is, the goals are to satisfy all of the clauses.

For each variable u_i, two operators are needed:

$$\overline{F(i)} \Rightarrow T(i)$$

$$\overline{T(i)} \Rightarrow F(i)$$

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 $u_i =$ 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:

$$T(i) \Rightarrow C(j)$$

$$F(i) \Rightarrow C(j)$$

Clearly, every $C(j)$ 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 it is *NP*-complete. □

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

Corollary 3.6 *PLANSAT$^1_{1+}$ is* NP*-complete.*

It is worth noting again at this point that a PLANSAT instance can be transformed to one with just positive postconditions if the postconditions between any two operators do not conflict. Recall that a postcondition "conflict" is when a postcondition of one operator is the negation of a postcondition of another operator.

3.3 Polynomial Propositional Planning

Theorem 3.7 *PLANSAT$^+_1$ is polynomial.*

Proof: The apparent difficulty is that some negative goals might need to be temporarily true to make some positive goals true or some negative goals false. However, because of the restrictions on the operators, it is sufficient to only consider plans that first make conditions true and then make conditions false.

Consider a sequence of operators in which the preconditions of each operator become true. Suppose that there are adjacent operators o_i and o_{i+1} such that o_i's postcondition is negative and o_{i+1}'s postcondition is positive. Let p_i be o_i's negative postcondition and p_{i+1} be o_{i+1}'s positive postcondition. If $p_i = p_{i+1}$, then o_i can be deleted because o_{i+1} reverses o_i's effect. If $p_i \neq p_{i+1}$, then o_i can be switched with o_{i+1} because their postconditions are independent of each other's preconditions. All of o_{i+1}'s preconditions are true after o_i, so leaving p_i positive cannot cause o_{i+1}'s preconditions to become false. Similarly, making p_{i+1} positive cannot cause o_i's preconditions to become false. Repeating these changes until there are no such adjacent operators will result in an equivalent sequence of operators that first makes conditions true and then makes conditions false.

Thus, if there is a solution, there is an intermediate state S that has the following properties:

S can be reached from the initial state $\mathcal{I}$ via operators with positive postconditions;

the positive goals $\mathcal{G}_+$ are a subset of S; and

$S \setminus \mathcal{G}_-$, i.e., a state that satisfies the negative goals, can be reached from S 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 $S \setminus \mathcal{G}_-$.

The algorithm for this problem depends on finding an intermediate state S that satisfies these properties.

Let TURNON be a subroutine that inputs a set of conditions X, and returns the maximal state $S \subseteq (\mathcal{P} \setminus X)$ that can be reached from the initial state $\mathcal{I}$. It is assumed that $X \cap \mathcal{I} = \emptyset$. TURNON can be implemented by the following algorithm.

TURNON(X)
  $S \leftarrow \mathcal{I}$
  repeat
      $temp \leftarrow S$
      for $o \in \mathcal{O}$ **do**
          if $(S \subset Result(S, o)) \wedge (X \cap Result(S, o) = \emptyset)$
              then $S \leftarrow Result(S, o)$
  until $S = temp$
  return S

TURNON simply keeps applying all operators until no more conditions can be added, making sure that no conditions in X are added. Adding a condition does not prevent the application of any other operator, so the maximal state S is unique. For n conditions and m operators, TURNON is $O(mn^2)$ (*Result* is $O(n)$ because an operator can have up to n preconditions).

Let TURNOFF be another subroutine that inputs a set of conditions S and returns the maximal state $S' \subseteq S$ such that $S \setminus \mathcal{G}_-$ can be reached from S'.

TURNOFF(S)
  $S' \leftarrow S \setminus \mathcal{G}_-$
  repeat
      $temp \leftarrow S'$
      for $o \in \mathcal{O}$ **do**
          if $(o_- \subseteq S) \wedge (Result(S' \cup o_-, o) = S')$
              then $S' \leftarrow S' \cup o_-$
  until $S' = temp$
  return S'

TURNOFF searches backward from $S \setminus \mathcal{G}_-$ to determine what operators can be used reach that state, as long as each operator only requires conditions in S. There is a unique maximal state S' because the backward search always adds conditions to S', which can only make more operators applicable, never fewer. For n conditions and m operators, TURNOFF is $O(mn^2)$.

SATISFY determines if a solution exists by iterating between TURNON and TURNOFF:

SATISFY
  $X \leftarrow \emptyset$
  loop
    $S \leftarrow$ TURNON(X)
    if $\mathcal{G}_+ \not\subseteq S$ **then return** reject
    $S' \leftarrow$ TURNOFF(S)
    if $S = S'$ **then return** accept
    $X \leftarrow X \cup (S \setminus S')$
    if $X \cap \mathcal{I} \neq \emptyset$ **then return** reject

In the first iteration S is set to TURNON($\emptyset$), i.e., S contains all the conditions that can be made true. If the positive goals are not true of S, then there is no way to achieve them, and the first if statement rejects the instance. Otherwise, S' is set to TURNOFF(S), i.e., S' is the maximum subset of S that can reach the negative goals. Because no other conditions can be made positive, S' is also the maximum subset of $\mathcal{P}$ that can reach the negative goals. If $S = S'$, then a solution exists, and the second if statement accepts the instance. If $S \neq S'$, then there must be some negative goals true of S that prevent the goal state from being reached. To achieve a goal state, these negative goals must never become true, and so are added to X. If some of these conditions are true of the initial state, the third if statement rejects the instance.

The next iteration is just like the first except that nothing in X is made true. This iteration might uncover additional conditions that, if made true, prevent the goal state from being reached. These conditions are added to X. In following iterations, either additional conditions are inserted in X, or one of the if statements classifies the instance. Because X grows monotonically and its size has an upper bound of n ($n = \mid \mathcal{P} \mid$), SATISFY performs at most n iterations. Both TURNON and TURNOFF are $O(mn^2)$ ($m = \mid \mathcal{O} \mid$), so SATISFY is $O(mn^3)$. $\square$

I show in a following section how Theorem 3.7 applies to the blocks-world.

Theorem 3.8 *For a constant g, PLANSAT^1 limited to g goals is polynomial.*

Proof: The following algorithm solves this problem by exhaustively searching backward from the goals.

Create a directed graph such that each vertex corresponds to a satisfiable set of g positive and negative conditions, and each edge (u, v) corresponds to u resulting from applying some operator o to v, i.e., u would be true of the next state if v were true of the previous state. Perform breadth-first search on the graph starting from the vertex corresponding to the goal. If a vertex true of the initial state is reached, then accept, otherwise reject.

This algorithm takes advantage of the following property of PLANSAT^1—a single goal cannot expand into multiple subgoals. Thus, if the g goals are reachable from the initial state, they can be traced to g or fewer (positive and negative) conditions true of the initial state. The graph constructed by the algorithm contains all possible links of any trace based on the operators. It is just a matter of graph search to find a path from the goals to a set of conditions true of the initial state.

Although the algorithm is polynomial, it is not a very good one. For n conditions, there are $2^g \binom{n}{g}$ sets of g conditions, which is $O(n^g)$. One operator can lead to $O(n^{2g})$ edges, e.g., an operator with no preconditions and n postconditions applies to all sets of g conditions and can lead to $\binom{n}{g}$ sets of g conditions. Breadth-first search is linear in the number of vertices and edges, so the above algorithm is $O(mn^{2g})$. □

Limiting operators to one precondition is crucial for this theorem. Otherwise, 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 transform the original set of goals onto a single "supergoal."

Theorem 3.9 *PLANSAT^0 is polynomial.*

Proof: The following algorithm solves PLANSAT^0 by working backwards from the goals.

Find an operator o that does not clobber any of the goals (for m operators and n conditions, this is $O(mn)$). Remove goals achieved by o from consideration. Also remove o from further consideration. Repeat the above until the remaining goals are true of the initial state (accept) or until no more appropriate operators can be found (reject). There are at most n goals, so this algorithm is $O(mn^2)$.

If a goal state can be reached, some operator must be the last operator, and it cannot reach the goal state if its postconditions are inconsistent with the goals. Whatever goals are achieved by this operator can be safely removed from consideration because the operator has no preconditions to worry about. Then the problem is simplified to a search for operators to achieve the remaining goals. □

3.4 Polynomial Planning without Explicit Operator Representations

The polynomial PLANSAT problems in this section have an interesting property. Suppose that the pre- and postconditions of the operators are not explicitly represented, but, instead, the *Result* function is supplied. In other words, a planning instance would have a set of operators and a function to simulate them, but no explicit representation of the structure of the operators.

It turns out that each problem is still polynomial. (Also, each *PSPACE*- and *NP*-complete PLANSAT problem remains *PSPACE*- and *NP*-complete.) For PLANSAT^1 restricted to g goals and PLANSAT^0, it is sufficient to note that the preconditions and postconditions of the operators can be recovered in polynomial time with appropriate use of the *Result* function.

For PLANSAT$_1^+$, note that TURNON only requires the *Result* function and TURNOFF only needs to know the negative postconditions of operators. For this problem, the negative postcondition of an operator o, if any, can be easily determined by looking at the result of $Result(\mathcal{P}, o)$.

Thus, there is a very striking difference between intractable and tractable PLANSAT problems. The tractable problems are easy even if the operators are implicitly represented. The intractable problems are hard even if the operators are explicitly represented.

3.5 The Blocks World

Theorem 3.7 can be used to show why finding non-optimal solutions to blocks-world instances is tractable.

Theorem 3.10 *The blocks-world problem can be reduced to PLANSAT$_1^+$.*

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

$\mathit{off}(i,j)$ $\quad B_i$ is *not* on top of B_j.

If B_i is on the table, then all $\mathit{off}(i,k)$ will be true. If B_i has a clear top, then all $\mathit{off}(k,i)$ will be true. If B_i is on top of B_j, then all $\mathit{off}(i,k)$ except for $\mathit{off}(i,j)$ will be true. off is used instead of the usual *on* in order to make the preconditions positive.

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

$$\bigwedge_{k=1}^{n} \mathit{off}(k,i) \wedge \bigwedge_{k=1}^{i-1} \mathit{off}(k,j) \wedge \bigwedge_{k=i+1}^{n} \mathit{off}(k,j) \Rightarrow \mathit{off}(i,j)$$

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:

$$\bigwedge_{k=1}^{n} \mathit{off}(k,i) \wedge \bigwedge_{k=1}^{n} \mathit{off}(i,k) \wedge \bigwedge_{k=1}^{n} \mathit{off}(k,j) \Rightarrow \overline{\mathit{off}(i,j)}$$

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

Since there are only $O(n^2)$ (i,j) combinations, only $O(n^2)$ 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 3.7, in a sense, explains why the blocks world is tractable.[10] $\Box$

[10]The SATISFY algorithm for Theorem 3.7 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.

An operator to move a block from one stack to another requires two postconditions. However, as noted above, moving a block from one stack to another has the same effect as two existing operators, each with one postcondition. Erol et al. [16] show that such "macro" operators can lead to shorter plans, and the problem remains tractable.

4 Complexity Results for PLANMIN

This section describes and demonstrates complexity results for PLANMIN, the decision problem of determining whether an instance of propositional STRIPS planning has a solution of k or fewer operators, where k is given as part of the input. First, I show that all the intractability results for PLANSAT transfer over to PLANMIN, then I consider the complexity of PLANMIN for restrictions that are tractable for PLANSAT.

4.1 Intractability Results for PLANMIN

Theorem 4.1
PLANMIN is PSPACE-*complete.*[11]
PLANMIN$_2^{2+}$ *is* PSPACE-*complete.*
PLANMIN$_1$ *is* PSPACE-*complete.*
PLANMIN^1 *is* PSPACE-*complete.*
PLANMIN$_+$ *is* NP-*complete.*[11]
PLANMIN$_{1+}^1$ *is* NP-*complete.*

Proof: Let $n = \mid \mathcal{P} \mid$. The first four problems are in *PSPACE* because the size of a state is limited by n, and a sequence of operators can be nondeterministicly chosen. The last two problems are in *NP* because the length of the shortest solution is bounded by n.

The *PSPACE*-hardness for the first four PLANMIN problems follows from the *PSPACE*-hardness of the corresponding PLANSAT problems. The PLANSAT problems can be reduced to the PLANMIN problems by setting $k = 2^n$.

The *NP*-hardness of the last two PLANMIN problems follows from the *NP*-hardness of the corresponding PLANSAT problems. The PLANSAT problems can be reduced to the PLANMIN problems by setting k to n. □

Theorem 4.2 *PLANMIN*$_1^+$ *is* NP-*complete.*

Proof: Because PLANSAT$_1^+$ is in *NP*, it follows that PLANMIN$_1^+$ is in *NP*. 3SAT can be reduced to PLANMIN$_1^+$.

Let $U = \{u_1, u_2, \ldots, u_m\}$ be the variables used in a instance of 3SAT. Let n be the number of clauses. An equivalent PLANMIN$_1^+$ instance can be constructed using the following types

[11]Also shown in Erol et al. [16].

of conditions.

$T(i)$	$u_i =$ true is selected.
$F(i)$	$u_i =$ false is selected.
$V(i)$	some value for u_i has been selected.
$C(j)$	The jth clause is satisfied.

The initial state and goals can be specified as:

$$\mathcal{I} = \emptyset$$
$$\mathcal{G} = V(1) \wedge V(2) \wedge \ldots \wedge V(m) \wedge C(1) \wedge C(2) \wedge \ldots \wedge C(n)$$

That is, the goals are to select a value for each variable and satisfy each of the clauses.

For each variable u_i, four operators are needed:

$$\Rightarrow T(i)$$

$$\Rightarrow F(i)$$

$$T(i) \Rightarrow V(i)$$

$$F(i) \Rightarrow V(i)$$

That is, there are operators to select values for u_i, and to ensure that a value has been selected for u_i. Although nothing prevents the selection of both true and false for a variable, it requires two operator applications to do so.

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:

$$T(i) \Rightarrow C(j)$$

$$F(i) \Rightarrow C(j)$$

If the 3SAT formula is satisfiable, then only one value needs to be selected for each variable (m operators), m operators are needed to set the $V(i)$'s, and n operators are needed to set the $C(i)$'s, for a total of $2m + n$ operators. If the 3SAT formula is not satisfiable, then both values must be selected for some variable to achieve the goals, which means more than $2m + n$ operators are needed. Thus, the 3SAT formula is satisfiable if and only if there is a plan of size $k = 2m + n$. □

All of the operators in the previous proof at most require one positive precondition and one positive postcondition, which leads to the following corollary.

Corollary 4.3 *PLANMIN$_{1+}^{1+}$ is* NP-*complete.*

This improves on the result in Erol et al. [16], who show that PLANMIN$_{+}^{1+}$ is *NP*-complete, but do not derive the minimum number of postconditions to obtain this result.

Theorem 4.4 (Erol et al.) *PLANMIN$_{3+}^{0}$ is* NP-*complete.*

Proof: Erol et al.'s reduction from minimum set covering to PLANMIN$^{1+}_{+}$ can be trivially modified to show that PLANMIN$^{0}_{3+}$ is *NP*-complete. In their reduction, the number of postconditions in an operator corresponds to the size of the cover. Minimum set covering is *NP*-complete for covers of size 3. □

Recall that PLANSAT^0 is polynomial. However, the above theorem leads to the following corollary.

Corollary 4.5 *PLANMIN^0 is* NP*-complete.*

There is one last intractable PLANMIN problem to demonstrate.

Theorem 4.6 *PLANMIN$^0_2$ is* NP*-complete.*

Proof: Because PLANSAT^0 is in *NP*, so is PLANMIN$^0_2$. 3SAT can be reduced to PLANMIN$^0_2$.

Using the same conditions as in the proof for PLANMIN$^+_1$, a PLANMIN$^0_2$ instance equivalent to a 3SAT instance can be constructed as follows. Use the following initial state and goals:

$$\mathcal{I} = \emptyset$$
$$\mathcal{G} = \overline{T(1)} \wedge \ldots \wedge \overline{T(m)} \wedge \overline{F(1)} \wedge \ldots \wedge \overline{F(m)} \wedge V(1) \wedge \ldots \wedge V(m) \wedge C(1) \wedge \ldots \wedge C(n)$$

That is, the goals are to deselect the values for each variable and satisfy each of the clauses.

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:

$$\Rightarrow T(i) \wedge C(j)$$

$$\Rightarrow F(i) \wedge C(j)$$

That is, to make $C(j)$ true, some value for some variable must be selected.

For each variable u_i, four operators are needed:

$$\Rightarrow \overline{T(i)}$$

$$\Rightarrow \overline{F(i)}$$

$$\Rightarrow T(i) \wedge V(i)$$

$$\Rightarrow F(i) \wedge V(i)$$

A value can be deselected at any time. To make V_i true, some value must be selected for u_i.

To make each $V(i)$ true, at least one value for each u_i must be selected (m operators). n operators are needed to make the $C(j)$'s true. At least m values will have been selected, so another m operators are needed to deselect the values. If the 3SAT formula is satisfiable, only m values need to be selected, and $2m + n$ operators need to be applied. If it is not, then both values must be selected for some variable to make all the $C(j)$'s true, which means more than $2m + n$ operators are needed. Thus, the 3SAT formula is satisfiable if and only if there is a plan of size $k = 2m + n$. □

4.2 Polynomial PLANMIN Problems

The above results do not leave much room for polynomial problems. Nevertheless, there are a few.

Theorem 4.7 *For a constant g, $PLANMIN^1$ limited to g goals is polynomial.*

Proof: The polynomial algorithm presented in the proof for Theorem 3.8 finds the shortest path from the initial state to a goal state. This is because each edge in the trace graph corresponds to achieving one set of g conditions from another set of g conditions via one operator, because the trace graph contains all such edges, and because breadth-first search finds the shortest path. □

Theorem 4.8 *$PLANMIN^0_{2+}$ is polynomial.*

Proof: This can be reduced to maximum matching of a graph, which has a polynomial algorithm [29]. A matching of a graph is a set of edges such that no two edges are incident to the same vertex.

Assuming that the goal is reachable, the reduction from a given PLANMIN^0_{2+} instance is as follows. Each operator that makes a negative goal true is removed. Each condition that is initially true is removed. Also, each condition that is not a positive goal is removed. Each remaining condition of the instance is mapped to a vertex. Each remaining operator with two postconditions is mapped to an edge (operators with one postcondition are ignored here). Now the maximum matching of this graph corresponds to applying the corresponding operators, where each operator achieves a disjoint pair of goals. Any remaining goals are achieved one at a time. If there are n positive goals to be achieved, and there are m operators in the maximum matching, then a goal state can be achieved using $n - m$ operators.

If there were a shorter plan, that would imply that more than m operators can achieve disjoint pairs of goals, which would imply that the maximum matching of the graph is larger than m, which is a contradiction. Therefore, the shortest plan must be of size $n - m$.

Determining reachability can be done in time linear in the number of conditions plus the number of operators. Micali and Vazivani's maximum matching algorithm [29] is $O(e\sqrt{v})$, where e is the number of edges, and v is the number of vertices. So for m operators and n conditions, checking for reachability is $O(m+n)$ and maximum matching is $O(m\sqrt{n})$, which implies that the total time is $O(n + m\sqrt{n})$. □

Theorem 4.9 *$PLANMIN^0_1$ is polynomial.*

Proof: Trivial. □

5 Extended Propositional STRIPS Planning

In this section, I define a augmented version of propositional STRIPS planning. After the definition, related work and a blocks-world example are discussed.

5.1 Definitions

An instance of *extended propositional STRIPS planning* is specified by a tuple $\langle \mathcal{P}, \Sigma, \mathcal{O}, \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$, where:

> $\mathcal{P}$ is a finite set of ground atomic formulas, called the *conditions*;
>
> Σ is a set of ground formulas, called the *domain theory*, that only uses ground atomic formulas from $\mathcal{P}$.
>
> $\mathcal{O}$ is a finite set of *operators*, where each operator has the form $Pre \Rightarrow Post$:
>
> > Pre is a satisfiable conjunction of positive and negative conditions, respectively called the *positive preconditions* (o^+) and the *negative preconditions* (o^-) of the operator;
> >
> > $Post$ is a satisfiable conjunction of positive and negative conditions, respectively called the *positive postconditions* (o_+) and the *negative postconditions* (o_-) of the operator;
> >
> > $\Sigma \cup \{Pre\}$ and $\Sigma \cup \{Post\}$ are consistent.
>
> $\mathcal{D}$, called the *default preference ordering*, is a total ordering of all positive and negative conditions, i.e., of all literals;
>
> $\mathcal{I} \subseteq \mathcal{P}$ is the *initial state*, where $\Sigma \cup \mathcal{I} \cup \{\overline{p} \mid p \in \mathcal{P} \setminus \mathcal{I}\}$ is consistent; and
>
> $\mathcal{G}$, called the *goals*, is a satisfiable conjunction of positive and negative conditions, respectively called the *positive goals* ($\mathcal{G}_+$) and the *negative goals* ($\mathcal{G}_-$), where $\Sigma \cup \{\mathcal{G}\}$ is consistent.

Again, a *state* is specified by a subset $S \subseteq \mathcal{P}$, indicating that $p \in \mathcal{P}$ is true in that state if and only if $p \in S$. A state S is *possible* if it is consistent with the domain theory Σ, i.e., if $\Sigma \cup S \cup \{\overline{p} \mid p \in \mathcal{P} \setminus S\}$ is consistent. The definition of an operator must be consistent with Σ; otherwise, impossible states can occur. Also, the initial state must be possible, and some possible state must satisfy the goals.

The default preference ordering $\mathcal{D}$ specifies which literals that are true before applying an operator are preferred to be true after applying an operator. The idea is that by default literals true of the previous state are true of the next state. However, if conflicts occur, i.e., inconsistency with the domain theory Σ, then $\mathcal{D}$ indicates which default is preferred. A precise definition is given below.

The consistency requirements simplify further definitions. Of course, determining whether a set of formulas is consistent can also be a hard problem. However, I shall only be considering tractable types of domain theories (definite Horn and Krom).

A finite sequence of operators $(o_1, o_2, \ldots, o_n)$ is a *solution* if $Result(\mathcal{I}, (o_1, o_2, \ldots, o_n))$ is a goal state, where $Result$ is defined as follows:

$$\begin{aligned}
Result(S, (\,)) &= S \\
Result(S, (o)) &= \begin{cases} Extend(S, \Sigma, o_+, o_-, \mathcal{D}) & \text{if } o^+ \subseteq S \text{ and } S \cap o^- = \emptyset \\ S & \text{otherwise} \end{cases} \\
Result(S, (o_1, o_2, &\ldots, o_n)) \\
&= Result(Result(S, (o_1)), (o_2, \ldots, o_n))
\end{aligned}$$

As before, o^+, o^-, o_+, and o_- respectively denote the sets of positive preconditions, negative preconditions, positive postconditions, and negative postconditions of operator o. Also, any operator can be applied to a state, but only has an effect if its preconditions are satisfied. An operator can appear multiple times in a sequence of operators. If its preconditions are satisfied, the next state is determined by the *Extend* function, which is defined as follows:

$$
\begin{array}{l}
\mathit{Extend}(S, \Sigma, S_1, S_2, (\ )) = S_1 \\
\mathit{Extend}(S, \Sigma, S_1, S_2, (p, l_1, \ldots, l_n)) = \\
\qquad \begin{cases}
\mathit{Extend}(S, \Sigma, S_1 \cup \{p\}, S_2, (l_1, \ldots, l_n)) \\
\qquad \text{if } p \in S \text{ and} \\
\qquad \Sigma \cup S_1 \cup \{p\} \cup \{\overline{q} \mid q \in S_2\} \text{ is consistent} \\
\mathit{Extend}(S, \Sigma, S_1, S_2 \cup \{p\}, (l_1, \ldots, l_n)) \\
\qquad \text{if } p \in S \text{ and} \\
\qquad \Sigma \cup S_1 \cup \{\overline{q} \mid q \in S_2\} \models \overline{p} \\
\mathit{Extend}(S, \Sigma, S_1, S_2, (l_1, \ldots, l_n)) \\
\qquad \text{if } p \notin S
\end{cases} \\
\mathit{Extend}(S, \Sigma, S_1, S_2, (\overline{p}, l_1, \ldots, l_n)) = \\
\qquad \begin{cases}
\mathit{Extend}(S, \Sigma, S_1, S_2 \cup \{p\}, (l_1, \ldots, l_n)) \\
\qquad \text{if } p \notin S \text{ and} \\
\qquad \Sigma \cup S_1 \cup \{\overline{p}\} \cup \{\overline{q} \mid q \in S_2\} \text{ is consistent} \\
\mathit{Extend}(S, \Sigma, S_1 \cup \{p\}, S_2, (l_1, \ldots, l_n)) \\
\qquad \text{if } p \notin S \text{ and} \\
\qquad \Sigma \cup S_1 \cup \{\overline{q} \mid q \in S_2\} \models p \\
\mathit{Extend}(S, \Sigma, S_1, S_2, (l_1, \ldots, l_n)) \\
\qquad \text{if } p \in S
\end{cases}
\end{array}
$$

If an operator is applied to a state S, and its preconditions are true, then $\mathit{Extend}(S, \Sigma, o_+, o_-, \mathcal{D})$ uses the postconditions and the default preference ordering $\mathcal{D}$ to decide whether literals true of the previous state S are true in the next state. Initially, literals appearing in the postconditions are assigned truth values, then the literals in $\mathcal{D}$ are considered in order. If a literal l is true in the previous state S and is consistent with the domain theory, the effects of the operator, and previously assigned literals, then l is assigned true in the next state; else if l is true in the previous state S, and the domain theory, the effects of the operator, and previously assigned literals imply the negation of l, then l is assigned false in the next state; else the decision is postponed.

The *Extend* function has the following properties. Given a possible state, *Extend* results in a possible state. Also, *Extend* makes a minimal number of changes from the previous state, i.e., the explicit effects of the operator plus sufficient changes to make the next state possible.

Propositional STRIPS planning is equivalent to extended propositional STRIPS planning with the domain theory $\Sigma = \emptyset$. Thus, extended propositional STRIPS planning is a strict generalization of propositional STRIPS planning.

Assuming that the default preference ordering is sufficiently expressive to resolve ambiguous results, first-order STRIPS planning can be polynomially reduced to extended propositional STRIPS planning under the same conditions as for propositional STRIPS planning with the following generalizations: the initial state, operators, and goals can contain non-literal formulas; all non-literal formulas in the initial state and in the postconditions of operators are true in all states, as is done in [28]; each quantified variable in a formula is limited to a polynomial number of values; and each formula is limited to a constant number of quantified variables.

5.2 Related Work

Extended propositional STRIPS planning is closely related to Ginsberg and Smith's [20] possible worlds approach to reasoning about actions. A possible state S in my framework corresponds to a "partial world" W (a set of first-order formulas) in theirs. Each partial world would also include the domain theory Σ as "protected" formulas. The postconditions of an operator o in my framework correspond to the consequences C of an action in theirs. They define a "possible world" as a maximal subset of $W \cup C$ that includes C, the protected formulas, and is consistent. The possible state resulting from the default preference ordering corresponds to a single possible world,[12] ignoring all other possible worlds. Ginsberg and Smith briefly discuss "prioritizing facts" in a manner similar to the default preference ordering so that a single possible world is preferred, but in their formalization, the resulting partial world is taken to be the intersection of all possible worlds.

Other approaches have dealt with the problem of determining the resulting state in a variety of ways. In TMM [13], when a set of time tokens conflict with a new time token, all the time tokens in the set are constrained to end before or begin after the new one. This conservative approach would possibly create more constraints than necessary to eliminate the contradiction. The plan net approach of Drummond [15] requires a "reconciliation set selection function" to choose among alternative ways to resolve inconsistency, but does not specify any constraints on its definition. Of course, in the situation calculus, this is where the infamous frame problem appears [22].

This problem can be recast as preferring one model over another, as in Shoham's preference logics [36]. Note that it is easy to map a possible state to its model. The result of the *Extend* function can then be formalized as follows:

Let S be a possible state, and M, its corresponding model. Let o be an operator whose preconditions are satisfied by S. Let S_1 and S_2 (with respective models M_1 and M_2) be two possible states consistent with o's postconditions. Let $\mathcal{D} = \{l_1, l_2, \ldots, l_n\}$ be a default preference ordering. Then $M_1 \sqsubset M_2$ (M_2 is preferred over M_1) if there exists an l_i such that:

for all l_j, $j < i$, $M_1 \models l_j$ if and only if $M_2 \models l_j$

$M \models l_i$, $M_1 \not\models l_i$, and $M_2 \models l_i$; or $M \not\models l_i$, $M_1 \models l_i$, and $M_2 \not\models l_i$.

Of course, the fact that *Extend* can be formalized in terms of model preference ordering does not make it reasonable. Doyle and Wellman [14] point out that any model preference ordering that is based on local ordering criteria (e.g., the default preference ordering) and that prefers one model over all others (e.g., the *Extend* function) generally violates some principle

[12] A minor(?) difference is that the formulas left out of a possible world are not negated in that possible world, while in a possible state, every condition is either true or false. To me, it looks like Ginsberg and Smith's definitions could be easily modified to take this into account.

of rational reasoning. In this case, extended propositional STRIPS planning violates the nondictatorship principle that no local criterion dominates all other criteria. For example, if l_1 is true of the previous state, is consistent with the postconditions and domain theory, and is first in the default preference ordering, then any possible state with l_1 true will be preferred as the next state over any possible state with l_1 false. Thus, Doyle and Wellman's result suggests that the default preference ordering is unreasonable for many domains, but it also suggests that either ambiguity must be tolerated (as in Ginsberg and Smith's possible world approach) or some other principle must be violated. Whatever alternative is chosen, it would be folly to expect much improvement in computational complexity.

5.3 Blocks-World Example

In the Sussman anomaly, there are three blocks A, 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 and B on C. Only one block at a time can be moved. The conditions, initial state, and goals can be represented as follows:

$$\begin{aligned}
\mathcal{P} &= \{on(A,B),\ on(A,C),\ on(B,A),\ on(B,C),\ on(C,A),\ on(C,B),\\
&\quad\ on(A,table),\ on(B,table),\ on(C,table),\ clear(A),\ clear(B),\ clear(C)\}\\
\mathcal{I} &= \{clear(C),\ on(C,A),\ on(A,table),\ clear(B),\ on(B,table)\}\\
\mathcal{M} &= \{on(A,B),\ on(B,C)\}\\
\mathcal{N} &= \{\}
\end{aligned}$$

The operators to stack and unstack blocks can be represented as:

$$\begin{aligned}
&clear(A) \wedge clear(B) \Rightarrow on(A,B)\\
&clear(A) \wedge clear(C) \Rightarrow on(A,C)\\
&clear(B) \wedge clear(A) \Rightarrow on(B,A)\\
&clear(B) \wedge clear(C) \Rightarrow on(B,C)\\
&clear(C) \wedge clear(A) \Rightarrow on(C,A)\\
&clear(C) \wedge clear(B) \Rightarrow on(C,B)\\
&clear(A) \Rightarrow on(A,table)\\
&clear(B) \Rightarrow on(B,table)\\
&clear(C) \Rightarrow on(C,table)
\end{aligned}$$

That is, if two blocks are clear, then stacking the first block on the second has the "direct" effect of the first block being on top of the second. If a block is clear, then unstacking the block has the "direct" effect of the block being on the table.

The other effects of these operators can be inferred from domain knowledge. Some effects can be inferred from the direct effects, e.g., if A is on B, then A cannot be on C or on the table. These are encoded in the domain theory Σ, which includes the following:

$$\begin{aligned}
&\overline{on(A,B)} \vee \overline{on(A,C)}\\
&\overline{on(A,B)} \vee \overline{on(A,table)}\\
&\overline{on(A,C)} \vee \overline{on(A,table)}\\
&\overline{on(B,A)} \vee \overline{on(C,A)}\\
&\overline{on(B,A)} \vee \overline{clear(A)}\\
&\overline{on(C,A)} \vee \overline{clear(A)}\\
&\ldots
\end{aligned}$$

$$
\begin{aligned}
&on(A,B) \vee on(A,C) \vee on(A,table)\\
&on(B,A) \vee on(B,C) \vee on(B,table)\\
&on(C,A) \vee on(C,B) \vee on(C,table)\\
&on(B,A) \vee on(C,A) \vee clear(A)\\
&on(A,B) \vee on(C,B) \vee clear(B)\\
&on(A,C) \vee on(B,C) \vee clear(C)
\end{aligned}
$$

Other effects depend on what was true in the previous state, e.g., if A was on C and is moved to B, then C is now clear. However, from the effect $on(A,B)$ of the operator and the domain theory formulas $\overline{on(A,B)} \vee \overline{on(A,C)}$ and $on(A,C) \vee on(B,C) \vee clear(C)$, it can only be inferred that either B is now on C or that C is now clear. Note though that in this encoding of the blocks-world, no *on* condition can indirectly become true after applying an operator, i.e., by default, false *on* conditions stay false. This domain knowledge can be encoded in the default preference ordering $\mathcal{D}$ by ordering negative *on* conditions before all others, as follows:

$$
\begin{aligned}
\mathcal{D} \;=\; (&\overline{on(A,B)},\ \overline{on(A,C)},\ \overline{on(A,table)},\ \ldots,\\
&on(A,B),\ on(A,C),\ on(A,table),\ \ldots,\\
&\overline{clear(A)},\ \overline{clear(B)},\ \overline{clear(C)},\\
&clear(A),\ clear(B),\ clear(C))
\end{aligned}
$$

As it happens, the ordering for the other conditions does not matter. Once negative *on* conditions are propagated from the previous state to the next state, all other changes are unambiguously implied by the domain theory.

Given the initial state of the Sussman anomaly, suppose that the operator to stack B on C is chosen:

$$clear(B) \wedge clear(C) \Rightarrow on(B,C)$$

Given the direct effect $on(B,C)$, the *Extend* function would be used to determine additional effects. Because negative *on* conditions are first in the default preference ordering $\mathcal{D}$, any negative *on* conditions true in the initial state and consistent with the operator's effect are determined to be true in the new state. Of course, $\overline{on(B,C)}$ is not consistent with $on(B,C)$, but the following literals are:

$$\overline{on(A,B)},\ \overline{on(A,C)},\ \overline{on(B,A)},\ \overline{on(C,B)},\ \overline{on(C,table)}$$

Positive *on* conditions are next in $\mathcal{D}$. $on(B,table)$ is not consistent with $on(B,C)$ and $\overline{on(B,C)} \vee \overline{on(B,table)}$ and so $\overline{on(B,table)}$ is inferred. The other positive *on* conditions—$on(C,A)$ and $on(A,table)$—remain true.

Negative *clear* conditions are next in $\mathcal{D}$. Only $\overline{clear(A)}$ is true of the initial state. It is also consistent with the effect of the operator, the indirect effects inferred so far, and the domain theory, so it is true of the new state.

Positive *clear* conditions are the last literals in $\mathcal{D}$. $clear(B)$ is consistent with the effects and the domain theory. $clear(C)$ is inconsistent with $on(B,C)$ and $\overline{on(B,C)} \vee \overline{clear(C)}$, so $\overline{clear(C)}$ is inferred.

6 Complexity Results for EPLANSAT

This section considers the computational complexity of extended propositional STRIPS planning assuming various restrictions on the domain theory and on operators. Let EPLANSAT be the decision problem of whether an instance of extended propositional STRIPS planning has a solution. As before, the notation $\text{EPLANSAT}^{\alpha}_{\beta}$ is used to denote EPLANSAT with restrictions α on the preconditions and restrictions β on the postconditions. For example, EPLANSAT^{0}_{1+} denotes EPLANSAT with operators limited to zero preconditions and one positive postcondition.

6.1 No Restrictions

Theorem 6.1 *EPLANSAT is* PSPACE-*complete.*

Proof: The results in Figure 1 from correspond to the case where the domain theory Σ is the empty set. Hence, EPLANSAT under the restriction that $\Sigma = \emptyset$ is *PSPACE*-complete, which implies that EPLANSAT is *PSPACE*-hard.

EPLANSAT is in *PSPACE* because 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*, EPLANSAT is also in *PSPACE*. Because it is also *PSPACE*-hard, EPLANSAT is *PSPACE*-complete. □

6.2 Definite Horn Domain Theories

A domain theory Σ is definite Horn if each formula in Σ is a definite Horn clause, i.e., a disjunction of literals containing exactly one positive literal.

Theorem 6.2 *EPLANSAT^{0}_{1+} restricted to definite Horn domain theories is* PSPACE-*complete.*

Proof: PLANSAT with operators limited to two positive preconditions and two postconditions is *PSPACE*-complete (Corollary 3.2). Each such operator can be converted into two operators, each with zero preconditions and one positive postcondition in combination with adding conditions to $\mathcal{P}$, adding definite Horn clauses to Σ, and imposing an appropriate preference ordering on literals.

Suppose there are m operators to convert. Suppose $p_{i1} \wedge p_{i2} \Rightarrow p_{i3} \wedge \overline{p_{i1}}$ is operator o_i to be converted (converting other kinds of operators will be similar).

Let $\Rightarrow pre(i)$ be the first operator, let $pre(i)$, $in(i)$, $post(i)$, and nil be new conditions, add the following definite Horn clauses to Σ:

$$\begin{array}{ll}
\overline{pre(i)} \vee \overline{p_{i1}} \vee \overline{p_{i2}} \vee in(i) & \\
\overline{pre(i)} \vee \overline{pre(j)} \vee nil & i \neq j \\
\overline{pre(i)} \vee \overline{in(j)} \vee nil & i \neq j \\
\overline{pre(i)} \vee \overline{post(j)} \vee nil & 1 \leq j \leq m
\end{array}$$

and impose the following orderings in the default preference ordering $\mathcal{D}$:

$$\begin{array}{ll} \overline{nil} \prec l & \text{for any other literal } l \\ p_{i1} \prec \overline{in(i)} & \\ p_{i2} \prec \overline{in(i)} & \end{array}$$

If p_{i1} and p_{i2} are true and nil false, then the result of applying $\Rightarrow pre(i)$ will be that $pre(i)$ and $in(i)$ are true, all other $pre(j)$ and $in(j)$ are false, and all $post(j)$ are false. The orderings ensure that nil remains false, and p_{i1} and p_{i2} remain true.

Let $\Rightarrow post(i)$ be the second operator, add the following definite Horn clauses to Σ:

$$\begin{array}{ll} \overline{post(i)} \vee \overline{in(i)} \vee p_{i3} & \\ \overline{post(i)} \vee \overline{in(i)} \vee \overline{p_{i1}} \vee nil & \\ \overline{post(i)} \vee \overline{post(j)} \vee nil & i \neq j \\ \overline{post(i)} \vee \overline{pre(j)} \vee nil & 1 \leq j \leq m \end{array}$$

and impose the following orderings in the default preference ordering $\mathcal{D}$:

$$\begin{array}{ll} \overline{nil} \prec l & \text{for any other literal } l \\ in(i) \prec \overline{p_{i3}} & \\ in(i) \prec p_{i1} & \end{array}$$

If $in(i)$ is true and nil false, then the result of applying $\Rightarrow post(i)$ will be that $post(i)$ and p_{i3} are true, p_{i1} is false, all other $post(j)$ are false, and all $pre(j)$ are false. The orderings ensure that nil remains false, and $in(i)$ remains true.

To have the effect of operator o_i, operator $\Rightarrow pre(i)$ can be followed by $\Rightarrow post(i)$. Also, any sequence of the converted operators will only have the effect of some sequence of original operators. Only a $\Rightarrow post(i)$ operator can affect the original conditions, and then only if $in(i)$ is true, which is possible only if $\Rightarrow pre(i)$ has been previously applied, if the preconditions of o_i are true, and if no intervening $\Rightarrow pre(j)$ has deleted $in(i)$.

Thus, starting with any initial state (new conditions initially false) and goals (new conditions not included), some sequence of the converted operators is a solution if and only if some sequence of the original operators is a solution. Thus, EPLANSAT$^0_{1+}$ restricted to definite Horn domain theories is *PSPACE*-hard. Because EPLANSAT is *PSPACE*-complete, this problem is also *PSPACE*-complete. □

6.3 Krom Domain Theories

A domain theory Σ is Krom if each formula in Σ is a Krom clause, i.e., a disjunction of two literals. Note that the default preference ordering does not matter because no "ambiguities" about indirect effects can occur for Krom domain theories, i.e., if one literal in a clause becomes false, the other literal must become true.

Theorem 6.3 *For any $k \geq 0$ and $g \geq 1$, the following problems can be polynomially reduced to each other:*

1. *EPLANSAT^k restricted to Krom domain theories g goals.*
2. *PLANSAT^k restricted to g goals; and*
3. *EPLANSAT$^{k+}_{1+}$ restricted to Krom domain theories and g goals;*

Proof: The first problem is clearly the most general of the three. Let o_i be an operator from some instance of the first problem, and let Σ be the instance's Krom domain theory.

Because Σ is Krom, the "indirect" effects of o_i will be exactly the same for any state satisfying o_i's preconditions. A new operator o'_i can be constructed that explicitly includes these effects in its postconditions. This can be done for every original operator, thus making Σ superfluous. Therefore, any instance of the first problem can be converted to an instance for the second problem.

For the third problem, a new operator o'_i can be constructed as follows. Suppose o_i has k preconditions. Let o'_i be the operator $pre(i,1) \wedge \ldots \wedge pre(i,k) \Rightarrow post(i)$, where $pre_(i,1)$, $\ldots$, $pre(i,k)$, and $post(i)$ are new literals. Note that there are k positive preconditions, and that there is exactly one positive postcondition. If l_j is the jth precondition of o_i, then add $\overline{l_j} \vee pre(i,j)$ and $l_j \vee \overline{pre(i,j)}$ to Σ. Thus, for any possible state, o'_i's preconditions are satisfied exactly when o_i's preconditions are satisfied. If l is a postcondition of o_i, then add $\overline{post(i)} \vee l$ to Σ. This ensures that the effects of o'_i include the effects of o_i. Also, add $\overline{post(i)} \vee \overline{post(j)}$ for $i \neq j$ to Σ. This ensures that o'_i does not have any more effects than o_i, except for any changes to new literals.

Now if $pre(i,j)$ conditions are added as needed to the initial state of the instance of the first problem so that the initial state is possible, then a sequence of converted operators is a solution if and only if the corresponding sequence of original operators is a solution. □

This theorem leads to the following corollaries derived from the complexity results for PLANSAT.

Corollary 6.4 *EPLANSAT$^{1+}_{1+}$ restricted to Krom domain theories is* PSPACE*-complete.*

Corollary 6.5 *EPLANSAT^1 restricted to Krom domain theories and g goals is polynomial.*

Corollary 6.6 *EPLANSAT^0 restricted to Krom domain theories is polynomial.*

7 Complexity Results for EPLANMIN

Let EPLANMIN be the problem of determining the existence of a solution of k operators or less for extended propositional STRIPS planning, where k is given as part of the input. For each *PSPACE*(*NP*)-complete EPLANSAT problem, it is also *PSPACE*(*NP*)-complete for EPLANMIN.

Theorem 7.1 *EPLANMIN is* PSPACE*-complete.*

EPLANMIN$^0_{1+}$ restricted to definite Horn domain theories is PSPACE*-complete.*

EPLANMIN$^{1+}_{1+}$ restricted to Krom domain theories is PSPACE*-complete.*

Proof: It is easy to set k so that the EPLANMIN problem is equivalent to the corresponding EPLANSAT problem. □

What remains are to consider the restrictions that are polynomial for EPLANSAT. The proof of Theorem 6.3 shows how to translate instances of extended propositional STRIPS planning with Krom domain theories to and from instances of propositional STRIPS planning. In each case, one operator in one instance is translated to an operator in the other instance with essentially identical effects. Thus, the proof of Theorem 6.3 also supports the translation of complexity results for PLANMIN to complexity results for EPLANMIN with Krom domain theories.

Theorem 7.2 *$EPLANMIN^0$ restricted to Krom domain theories is* NP*-complete, even if operators are limited to one positive postcondition.*

Proof: Follows from *NP*-completeness of $PLANMIN^0$. □

Theorem 7.3 *$EPLANMIN^1$ restricted to Krom domain theories and g goals is polynomial.*

Proof: Follows from polynomial result for $PLANMIN^1$ restricted to g goals. □

8 Conclusion

This analysis shows that extremely severe restrictions on both the operators and the domain theory are required to guarantee tractability or even NP-completeness for planning problems. One must be careful, however, concerning the implications of these results.

Work on reactive and anytime planning systems is partly motivated by the complexity of planning. However, this motivation is somewhat misguided because the complexity arises from the properties of the problem, not from the properties of any particular algorithm that solves the problem. In other words, the complexity results specify how hard it is to find a sequence of actions that accomplish a set of goals, but are completely neutral to how the sequence of actions is generated. Whether or not a system is reactive, anytime, or sound and complete, it is equally hard to achieve goals by performing actions. This in no way invalidates work on different types of planning algorithms, just that they must be put into the proper context.

Nevertheless, many successful planning systems attest to the fact that planning is indeed possible and practical for many domains. What then accounts for the large gap between the theoretical hardness of planning and its practical application? Perhaps a large part of the answer is that the complexity analysis only considers properties of planning problems that are domain-independent, and so have some chance of being generally applicable. However, the analysis strongly suggests that there is no such thing as a set of generally-applicable domain-independent properties that lead to efficient planning. A pessimistic view is that practical planning domains are efficient for domain-dependent reasons, which in turn demand domain-dependent analyses and planning algorithms. More optimistically, it is possible that specific classes of planning domains might be efficiently solvable for similar reasons. In any case, the key question is how much can domain-independent planning live up to its name.

References

[1] J. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning.* Morgan Kaufmann, San Mateo, California, 1990.

[2] F. Bacchus and Q. Yang. The downward refinement property. In *Proc. Twelfth Int. Joint Conf. on Artificial Intelligence*, pages 286–292, Sydney, 1991.

[3] C. Bäckström and I. Klein. Parallel non-binary planning in polynomial time. In *Proc. Twelfth Int. Joint Conf. on Artificial Intelligence*, pages 268–273, Sydney, 1991.

[4] C. Bäckström and B. Nebel. Complexity results for SAS+ planning. In *Proc. Thirteenth Int. Joint Conf. on Artificial Intelligence*, Chambery, France, 1993. To appear.

[5] T. Bylander. Complexity results for planning. In *Proc. Twelfth Int. Joint Conf. on Artificial Intelligence*, pages 274–279, Sydney, 1991.

[6] T. Bylander. Complexity results for extended planning. In *Proc. First Int. Conf. on AI Planning Systems*, pages 20–27, College Park, Maryland, 1992.

[7] T. Bylander. Complexity results for serial decomposability. In *Proc. Tenth National Conf. on Artificial Intelligence*, pages 729–734, San Jose, California, 1992.

[8] T. Bylander. An average case analysis of planning. In *Proc. Eleventh National Conf. on Artificial Intelligence*, pages 480–485, Washington, DC, 1993.

[9] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987. Also appears in [1].

[10] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proc. Twelfth Int. Joint Conf. on Artificial Intelligence*, pages 331–337, Sydney, 1991.

[11] S. V. Chenoweth. On the NP-hardness of blocks world. In *Proc. Ninth National Conf. on Artificial Intelligence*, pages 623–628, Anaheim, California, 1991.

[12] T. Dean and M. Boddy. Reasoning about partially ordered events. *Artificial Intelligence*, 36(3):375–399, 1988.

[13] T. L. Dean and D. V. McDermott. Temporal data base management. *Artificial Intelligence*, 32(1):1–55, 1987. Also appears in [1].

[14] J. Doyle and M. P. Wellman. Impediments to universal preference-based default theories. *Artificial Intelligence*, 49:97–128, 1991.

[15] M. E. Drummond. A representation of action and belief for automatic planning systems. In *Reasoning about Actions and Plans: Proc. of the 1986 Workshop*, pages 189–211, Timberline, Oregon, 1987.

[16] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability, and undecidability results for domain-independent planning. Technical report, Dept. of Computer Science, Univ. of Maryland, College Park, Maryland, 1991.

[17] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971. Also appears in [1].

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman, New York, 1979.

[19] M. P. Georgeff. Planning. *Annual Review of Computer Science*, 2:349–400, 1987. Also appears in [1].

[20] M. L. Ginsberg and D. E. Smith. Reasoning about actions I: A possible worlds approach. *Artificial Intelligence*, 35(2):165–195, 1988.

[21] N. Gupta and D. S. Nau. Complexity results for blocks-world planning. In *Proc. Ninth National Conf. on Artificial Intelligence*, pages 629–633, Anaheim, California, 1991.

[22] P. J. Hayes. The frame problem and related problems in artificial intelligence. In A. Elithorn and D. Jones, editors, *Artificial and Human Thinking*, pages 45–59. Jossey-Bass, San Francisco, 1973.

[23] J. Hendler, A. Tate, and M. Drummond. AI planning: Systems and techniques. *AI Magazine*, 11(2):61–77, 1990.

[24] R. E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985.

[25] R. E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987. Also appears in [1].

[26] M. R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 13(1):15–20, 1967.

[27] H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3(1):78–93, 1987.

[28] V. Lifschitz. On the semantics of STRIPS. In *Reasoning about Actions and Plans: Proc. of the 1986 Workshop*, pages 1–9, Timberline, Oregon, 1987. Also appears in [1].

[29] S. Micali and V. V. Vazivani. An $O(\sqrt{|v|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *Proc. 21st IEEE Annual Symposium on Foundations of Computer Science*, pages 17–27, Syracuse, New York, 1980.

[30] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[31] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. Tenth National Conf. on Artificial Intelligence*, pages 459–465, San Jose, California, 1992.

[32] R. Musick and S. Russell. How long will it take? In *Proc. Tenth National Conf. on Artificial Intelligence*, pages 466–471, San Jose, California, 1992.

[33] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga, Palo Alto, California, 1980.

[34] D. Ratner and M. Warmuth. Finding a shortest solution for the $n \times n$ extension of the 15-*puzzle* is intractable. In *Proc. Fifth National Conf. on Artificial Intelligence*, pages 168–172, Philadelphia, 1986.

[35] W. J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. Computer and System Sciences*, 4:177–192, 1970.

[36] Y. Shoham. *Reasoning about Change.* MIT Press, Cambridge, Massachusetts, 1988.