Complexity, Decidability and Undecidability Results for Domain-Independent Planning

Kutluhan Erol, Dana S. Nau, and V.S. Subrahmanian

email: {kutluhan,nau,vs} @cs.umd.edu

http: //www.cs.umd.edu/{ ˜kutluhan, ˜nau, ˜vs}

Department of Computer Science, Institute for Systems Research, and Institute for Advanced Computer Studies. University of Maryland College Park, Maryland 20742, U.S.A.

Abstract

In this paper, we examine how the complexity of domain-independent planning with STRIPS-style operators depends on the nature of the planning operators.

We show conditions under which planning is decidable and undecidable. Our results on this topic solve an open problem posed by Chapman [5], and clear up some difficulties with his undecidability theorems.

For those cases where planning is decidable, we explain how the time complexity varies depending on a wide variety of conditions:

– whether or not function symbols are allowed;

– whether or not delete lists are allowed;

– whether or not negative preconditions are allowed;

– whether or not the predicates are restricted to be propositional (i.e., 0-ary);

– whether the planning operators are given as part of the input to the planning problem, or instead are fixed in advance.

– whether or not the operators can have conditional effects.

1 Introduction

Much planning research has been motivated, in one way or another, by the difficulty of producing complete and correct plans. For example, techniques such as abstraction [27,6,24,31] and task reduction [28,6,31] were developed in an effort to make planning more efficient, and concepts such as deleted-condition interactions were developed to describe situations which make planning difficult.

Despite the acknowledged difficulty of planning, it is only recently that researchers have begun to examine the computational complexity of planning problems and the reasons for that complexity [5,3,16,17,22,20]. This research has yielded some surprising results. For example, Gupta and Nau [16,17] have shown that contrary to prior expectations, deleted-condition interactions are easy to handle in blocks-world planning.

Pednault [25] suggests that since planning is intractable in general, researchers should try to identify constraints that will lead to efficient planning. The current paper addresses this goal, by examining how the complexity of domain-independent planning depends on the nature of the planning operators.

We consider planning problems in which the current state is a set of ground atoms, and each planning operator is a STRIPS-style operator consisting of three lists of atoms: a precondition list, an add list, and a delete list. Our results can be summarized as follows:

(i) If function symbols are allowed, then determining, in general, whether a plan exists[1] is *undecidable* (more specifically, semidecidable).[2] This is true even if we have no delete lists and the precondition list of each operator contains at most one (non-negated) atom. If no function symbols are allowed and only finitely many constant symbols are allowed, then plan existence is *decidable*, regardless of the presence or absence of delete lists and/or negated preconditions.

Even when function symbols are present, plan existence is decidable if the planning domains being considered have no delete lists, no negated atoms occur in the precondition list, and the domains satisfy certain acyclicity and boundedness properties.

(ii) When there are no function symbols and only finitely many constant symbols (so that planning is decidable), the computational complexity varies from constant time to EXPSPACE-complete, depending on the following conditions:

---

[1] The formal definition of this problem appears in Section 2.

[2] We use "decidable" and "undecidable" interchangably with "recursive" and "recursively enumerable," respectively.

- whether or not we allow delete lists and/or negative preconditions,
- whether or not we restrict the predicates to be propositional (i.e., 0-ary),
- whether we fix the planning operators in advance, or give them as part of the input.

(iii) We have solved an open problem stated by Chapman in [5]: whether or not planning is undecidable when the language contains infinitely many constants but the initial state is finite. In particular, this problem is decidable in the case where the planning operators have no negative preconditions and no delete lists. If the planning operators are allowed to have negative preconditions and/or delete lists, then the problem is undecidable.

(iv) Chapman's Second Undecidability Theorem states that "planning is undecidable even with a finite initial situation if the action representation is extended to represent actions whose effects are a function of their input situation" [5], i.e., if the language contains function symbols and infinitely many constants. Our results show that even with a number of additional restrictions, planning is still undecidable.

   We also correct a misimpression about this theorem, which has been thought by some researchers [26,11] to refer to operators that have conditional effects. It does not—and our decidability and complexity results are unaffected by whether or not the operators have conditional effects.

(v) Chapman [5] and Dean and Boddy [8] studied planning with conditional operators, and showed that the problem of deciding whether a proposition is necessarily true after a partially ordered plan (a.k.a modal truth criterion) is NP-hard in the presence of conditional operators. The same problem can be solved in polynomial time when conditional operators are not allowed, and this led researchers to believe that planning with conditional operators is harder than planning with regular STRIPS operators. However, our results show that, contrary to the expectations, conditional operators do not affect the complexity of plan existence, nor the complexity of plan optimality problems.

The rest of this paper is organized as follows. Section 2 contains the basic definitions. Section 3 discusses the decidability and undecidability results. Section 4 discusses the complexity results. Section 5 discusses the related work. Section 6 contains concluding remarks, and discusses future research directions. For a more extensive treatment, including all of the mathematical details, see [10].

2   Preliminaries

Researchers in planning have long been interested in planning with STRIPS-style operators, and this interest still continues [3,5,16,22,20]. In the original

STRIPS planner [13], the planning operators' precondition lists, add lists, and delete lists were allowed to contain arbitrary well-formed formulas in first-order logic. However, there were a number of problems with this formulation, such as the difficulty of providing a well-defined semantics for it [19]. Thus, in subsequent work, researchers have placed some restrictions on the nature of the planning operators [24]. Typically, the precondition lists, add lists and delete lists contain only atoms, and the goal is a conjunct of ground or existentially quantified atoms. Our definitions below are in accordance with such commonly accepted formulations.

2.1 Basic Definitions

If $\mathcal{L}$ is a first-order language, then a *state* is a set of ground atoms in $\mathcal{L}$. Intuitively, a state tells us which ground atoms are currently true: if a ground atom A is in state S, then A is true in state S, and if $B \notin S$, then B is false in state S. Thus, a state is simply an Herbrand interpretation (cf. Shoenfield [29]) for the language $\mathcal{L}$, and hence each formula of first-order logic is either satisfied or not satisfied in S according to the usual first-order logic definition of satisfaction.

We use STRIPS-style planning operators similar to those used by Nilsson [24]. A *planning operator* α is a 4-tuple $(\text{Name}(\alpha), \text{Pre}(\alpha), \text{Add}(\alpha), \text{Del}(\alpha))$, where

(i) $\text{Name}(\alpha)$ is a syntactic expression of the form $\alpha(X_1, \ldots, X_n)$ where each X_i is a variable symbol of $\mathcal{L}$;

(ii) $\text{Pre}(\alpha)$ is a finite set of literals (i.e., atoms and negated atoms), called the *precondition list* of α, whose variables are all from the set $\{X_1, \ldots, X_n\}$;

(iii) $\text{Add}(\alpha)$ and $\text{Del}(\alpha)$ are both finite sets of atoms (possibly non-ground) whose variables are taken from the set $\{X_1, \ldots, X_n\}$. $\text{Add}(\alpha)$ is called the *add list* of α, and $\text{Del}(\alpha)$ is called the *delete list* of α.

Observe that negated atoms are allowed in the precondition list, but not in the add and delete lists.

A *planning domain* is a pair $\mathbf{P} = (S_0, \mathcal{O})$, where S_0 is a state called the *initial state*, and $\mathcal{O}$ is a finite set of planning operators. The *language* of $\mathbf{P}$ is the first-order language $\mathcal{L}$ generated by the constant, function, predicate, and variable symbols appearing in $\mathbf{P}$, along with an infinite number of additional variable symbols.

A *goal* is a conjunction of atoms which is existentially closed (i.e., the variables, if any, are existentially quantified). A *planning problem instance* is a triple $\mathbf{P} = (S_0, \mathcal{O}, G)$, where $(S_0, \mathcal{O})$ is a planning domain and G is a goal.

Let $\mathbf{P} = (S_0, \mathcal{O})$ be a planning domain, α be an operator in $\mathcal{O}$ whose name is $\alpha(X_1, \ldots, X_n)$, and θ be a substitution that assigns ground terms to each $X_i, 1 \leq i \leq n$. Suppose that the following conditions hold for states S and S':

$\{A\theta : A$ is a positive literal in $\mathrm{Pre}(\alpha)\} \subseteq S$;
$\{B\theta : \neg B$ is a negative literal in $\mathrm{Pre}(\alpha)\} \cap S = \emptyset$;
$S' = (S - (\mathrm{Del}(\alpha)\theta)) \cup (\mathrm{Add}(\alpha)\theta)$.

Then we say that α is θ-executable in state S, resulting in state S'. This is denoted symbolically as

$$S \xRightarrow{\alpha, \theta} S'.$$

Suppose $\mathbf{P} = (S_0, \mathcal{O})$ is a planning domain and G is a goal. A plan that achieves G is a sequence $S_0, \ldots, S_n$ of states, a sequence $\alpha_1, \ldots, \alpha_n$ of planning operators, and a sequence $\theta_1, \ldots, \theta_n$ of substitutions such that

$$S_0 \xRightarrow{\alpha_1, \theta_1} S_1 \xRightarrow{\alpha_2, \theta_2} S_2 \cdots \xRightarrow{\alpha_n, \theta_n} S_n \tag{1}$$

and G is satisfied by S_n, i.e. there exists a ground instance of G that is true in S_n. The length of the above plan is n.

We now define two decision problems:

– PLAN EXISTENCE is the problem, "Given a planning problem instance $\mathbf{P} = (S_0, \mathcal{O}, G)$, is there a plan in $\mathbf{P}$ that achieves G?"
– PLAN LENGTH is the problem, "Given a planning problem instance $\mathbf{P} = (S_0, \mathcal{O}, G)$ and an integer k encoded in binary, is there a plan in $\mathbf{P}$ of length k or less that achieves G?"

In the definition of PLAN LENGTH, what really interests us is not whether there is a plan of length k or less, but finding the shortest plan. This problem is at least as difficult as PLAN LENGTH, and in some cases harder. For example, in the Towers of Hanoi problem [1] and certain generalizations of it [15], the length of the shortest plan can be found in low-order polynomial time—but actually producing this plan requires exponential time and space, since the plan has exponential length. The definition of PLAN LENGTH follows the standard procedure for converting optimization problems into yes/no decision problems (cf. [14, pp. 115–117]).

2.2 Special-Case Definitions

2.2.1 Acyclicity and Boundedness

In this section, we introduce various restrictions on the structure of planning domains and/or goals which guarantee that the planning problem is decidable, even if function symbols are allowed in the language.

A *level mapping* for a language L is a mapping $\ell : AT(L) \to \mathbf{N}$ where $AT(L)$ is the set of ground atoms in language L and $\mathbf{N}$ is the set of natural numbers.

Intuitively, a level mapping partitions the set of all ground atoms into a collection of "levels." In the same vein, a predicate level mapping, defined below, partitions the set of predicate symbols into a collection of levels.

A *predicate level mapping* for L is a mapping $\sharp : Pred(L) \to \mathbf{N}$ where $Pred(L)$ is the set of predicate symbols in language L.

Suppose $\mathbf{P} = (S_0, \mathcal{O})$ is a planning domain in which no operator has negative preconditions or delete lists. Intuitively, $\mathbf{P}$ is acyclic if achieving the atoms (resp. predicates) in the add list of any operator in $\mathbf{P}$ only depend on having to achieve atoms (resp. predicates) at a strictly lower-level. Formally, $\mathbf{P}$ is said to be *atomically acyclic* iff there exists a level mapping ℓ such that for any ground instance α of operators in $\mathbf{P}$, it is the case that $\ell(A) > \ell(B)$ for all $A \in Add(\alpha)$ and $B \in Pre(\alpha)$. $\mathbf{P}$ is said to be *predicate acyclic* iff there exists a predicate level mapping $\sharp$ such that for all operators α in $\mathbf{P}$, it is the case that $\sharp(p) > \sharp(q)$ for all predicates p occurring in $Add(\alpha)$ and all predicates q occurring in $Pre(\alpha)$.

Sometimes, an atom in the add list of a ground instance of an operator may not be achievable. If the only way that this can happen is because there is an unachievable atom at a lower level in the precondition of the same ground instance of that operator, then the planning domain is said to be weakly recurrent.

Formally, a planning domain $\mathbf{P} = (S_0, \mathcal{O})$ is *weakly recurrent* iff there exists a level mapping ℓ such that for every ground instance α of an operator in $\mathcal{O}$, if $A \in Add(\alpha)$ is such that there is no plan to achieve A from $\mathbf{P}$, then there is a $B_i \in Pre(\alpha)$ such that there is no plan to achieve B_i from $\mathbf{P}$ and $\ell(A) > \ell(B_i)$.

For some goals, there will exist an upper bound on the levels of all ground instances of the goal. When this happens, we say that the goal is *bounded*, as defined formally below. When a goal is bounded in a weakly recurrent planning domain, this allows us to infer a bound on the length of a plan to achieve the goal, thus causing planning for such goals in such domains to be decidable.

Suppose $G = (\exists)(A_1 \& \ldots \& A_n)$ is a goal. Let $Grd(G)$ denote the set of all ground instances of the quantifier-free conjunction $(A_1 \& \ldots \& A_n)$. G is *bounded* w.r.t. a level mapping ℓ iff there is an integer b such that for every ground instance $(A_1 \& \ldots \& A_n)\theta$ in $grd(G)$, it is the case that $\ell(A_i) < b$.

2.2.2 Conditional Planning Operators

Several researchers [5,8,26,25] have been interested in actions whose effects depend on the input situation. The following formulation of conditional planning operators is due to Dean and Boddy [8]. A *conditional operator* α is a finite set $\{t_1, t_2, \ldots, t_n\}$, where each t_i is a triple of the form $\langle \mathrm{Pre}_i, \mathrm{Del}_i, \mathrm{Add}_i \rangle$. Pre_i, Del_i, and Add_i correspond to the precondition list, delete list and add list associated with the i'th triple, respectively.

Suppose α is a conditional operator, θ is a ground substitution for the variables appearing in α, S is a state, $I = \{i : S \text{ satisfies } \mathrm{Pre}_i\theta\}$, and

$$S' = (S - \bigcup_{i \in I} \mathrm{Del}_i\theta) \cup \bigcup_{i \in I} \mathrm{Add}_i\theta.$$

Then we say that α is θ-*executable* in state S, *resulting* in state S'. This is denoted as $S \stackrel{\alpha,\theta}{\Longrightarrow} S'$. Note that *all* triples with satisfied preconditions contribute to the output state.

Our results are independent of whether we use conditional operators such as the ones defined above, or the ordinary STRIPS-style planning operators in Section 2.1.

3 Decidability and Undecidability Results

In [10], we have proved theorems that show:

  (i) how to transform a planning domain with delete lists into one without delete lists when $\mathcal{L}$ contains no function symbols;

 (ii) how to transform, in polynomial time, a planning domain without delete lists and without negative preconditions into a logic program such that for all goals G, the goal G is achievable from the planning domain iff the logical query that G represents is provable from the corresponding logic program;

(iii) how to transform, in polynomial time, a logic program into an equivalent planning domain in which each operator has no negative preconditions and no delete lists.

Table 1
Decidability of domain-independent planning.[α]

Allow function symbols?	Allow infinitely many constant symbols?[β]	infinite initial states?[β]	Allow delete lists and/or negated preconditions?	PLAN EXISTENCE (telling if a plan exists)
yes	yes/no	yes/no	yes/no/no[γ]	semidecidable
	no	no	no[δ]	decidable
no	yes	yes	yes/no	semidecidable
		no	yes	semidecidable
			no	decidable
	no	no[ε]	yes/no	decidable

[α]All results are independent of whether the operators are given as part of the input or fixed in advance, and whether or not the operators are allowed to have conditional effects.

[β]First-order languages are usually assumed to contain only finitely many constant symbols, and states are usually assumed to contain only finitely many atoms. However, for comparison with Chapman's [5] results, we also consider the cases where they are infinite.

[γ]No operator has more than one precondition.

[δ]With acyclicity and boundedness restrictions as described in Section 2.2.1.

[ε]In this case, the other restrictions ensure that the initial state will always be finite.

The above results establish that logic programming is essentially the same as planning without delete lists. This equivalence allows us to transport many results from logic programming to planning, leading to a number of decidability and undecidability results. Our decidability and undecidability results are summarized in Table 1 (for their details, see [10]). If we use the conventional definitions of a first-order language (i.e., the language contains only finitely many constant symbols), then whether or not PLAN EXISTENCE is decidable depends largely on whether or not function symbols are allowed:

(i) If the language is allowed to contain function symbols (and hence infinitely many ground terms), then, in general, PLAN EXISTENCE is undecidable, regardless of whether or not the operators have delete lists, negative preconditions, or more than one precondition.

(ii) When certain syntactic (predicate and atomic acyclicity) and semantic properties (weak recurrence) are satisfied by planning domains (even those containing function symbols) in which there are no delete lists or negative preconditions, then plan existence for bounded goals is decidable.

(iii) If the language does not contain function symbols (and hence has only finitely many ground terms), then PLAN EXISTENCE is decidable, regardless of whether or not the planning operators have negative preconditions, delete lists, or more than one precondition.

Whether the planning operators are fixed in advance or given as part of the input, and whether or not they are conditional, does not affect these results.

For comparison with Chapman's [5] results, Table 1 also includes decidability and undecidability results for the cases where we allow infinitely many constant symbols, infinite initial states, and operators with conditional effects. These results relate to Chapman's work as follows:

– Our results solve an open problem stated by Chapman in [5]: whether or not planning is undecidable when the language contains infinitely many constants but the initial state is finite. In particular, our results show that this problem is decidable in the case where the planning operators have no negative preconditions and no delete lists. If the planning operators are allowed to have negative preconditions and/or delete lists, then the problem is undecidable.
– Chapman's Second Undecidability Theorem states that "planning is undecidable even with a finite initial situation if the action representation is extended to represent actions whose effects are a function of their input situation" [5], i.e., if the language contains function symbols and infinitely many constants.[3] Our results subsume this theorem, by showing that even with a number of additional restrictions, planning is still undecidable.

4   Complexity Results

Based on various syntactic criteria on what planning operators are allowed to look like, we have developed a comprehensive theory of the complexity of planning. The results are summarized in Table 2; for details see [10]. When there are no function symbols and only finitely many constant symbols (so that planning is decidable), the computational complexity varies from constant time to EXPSPACE-complete, depending on a wide variety of conditions:

– whether or not delete lists are allowed;
– whether or not negative preconditions are allowed;
– whether or not the predicates are restricted to be propositional (i.e., 0-ary);

---

[3] The phrase "actions whose effects are a function of their input situation" has been thought by some researchers [26,11] to refer to conditional operators. However, a careful examination of Chapman's proof makes it clear that he is referring to the case where the planning operators are allowed to contain function symbols.

Table 2
Complexity of domain-independent planning.[α]

Language restrictions	How the operators are given	Allow delete lists?	Allow negated preconditions?	PLAN EXISTENCE (telling if a plan exists)	PLAN LENGTH (if there is a plan of length $\leq k$)
datalog (no function symbols, and only finitely many constant symbols)	given in the input	yes	yes/no	EXPSPACE-comp.	NEXPTIME-comp.
		no	yes	NEXPTIME-comp.	NEXPTIME-comp.
			no	EXPTIME-comp.	NEXPTIME-comp.
			no[β]	PSPACE-complete	PSPACE-comp.
	fixed in advance	yes	yes/no	PSPACE [δ]	PSPACE [δ]
		no	yes	NP [δ]	NP [δ]
			no	P	NP [δ]
			no[β]	NLOGSPACE	NP
propositional (all predicates are 0-ary)	given in the input	yes	yes/no	PSPACE-complete[ε]	PSPACE-comp.
		no	yes	NP-complete[ε]	NP-complete
			no	P [ε]	NP-complete
			no[β]/no[γ]	NLOGSPACE-comp.	NP-complete
	fixed in advance	yes/no	yes/no	constant time	constant time

[α]All results are independent of whether or not the operators are allowed to have conditional effects.

[β]No operator has more than one precondition.

[γ]Every operator with more than one precondition is the composition of other operators.

[δ]With PSPACE- or NP-completeness for some sets of operators.

[ε]Results due to Bylander [3].

– whether the planning operators are given as part of the input to the planning problem, or instead are fixed in advance.

Below, we summarize how and why our parameters affect the complexity of planning:

(i) If no restrictions are put on the planning domain **P**, any operator instance might need to appear many times in the same plan, forcing us to search through all the states, which are double exponential in number. Since the size of any state is at most exponential, PLAN EXISTENCE can be solved

in EXPSPACE.

(ii) If the planning operators are restricted to have no delete lists, then any predicate instance asserted remains true throughout the plan, hence no operator instance needs to appear in the same plan twice. Since the number of operator instances is exponential, this reduces the complexity of PLAN EXISTENCE to NEXPTIME.

(iii) If the planning operators are further restricted to have no negative preconditions, then no operator can ever clobber another. Thus the order of the operators in the plan does not matter, and the complexity of PLAN EXISTENCE reduces to EXPTIME.

(iv) In spite of the restrictions above, PLAN LENGTH remains NEXPTIME. Since we try to find a plan of length at most k, which operator instances we choose, and how we order them makes a difference.

(v) If each planning operator is restricted to have at most one precondition, then we can do backward search, and since each operator has at most one precondition, the number of the subgoals does not increase. Thus both PLAN EXISTENCE and PLAN LENGTH with these restrictions can be solved in PSPACE.

(vi) The previous arguments also hold for propositional planning, with the exception of the anomaly in the unrestricted case for PLAN LENGTH, which we discuss later on. As a result of restricting predicates to be 0-ary, the number of operator instances and the size of each state reduce to polynomial from exponential. Hence in general, the complexity results for propositional planning are one level lower than the complexity results with datalog operators. We can get the same amount of reduction in complexity by placing a constant bound on the arity of predicates and the number of variables in each operator. Propositional planning corresponds to the case where the bound is zero.

(vii) When the operator set is fixed in advance, the arity of predicates and the number of variables in each operator are bound by a constant, thus the complexity of planning with a fixed set of operators is the same as complexity of propositional planning. Our results on planning with a fixed set of operators reveal that for any given planning domain that can be described with STRIPS operators, the complexity of planning is at most in PSPACE, and that there exists such domains for which planning is PSPACE-complete.

Examination of Table 2 reveals several interesting properties:

(i) If the planning operators are extended to allow conditional effects, this does not affect our results. This contradicts a widespread belief that planning with conditional operators is harder than planning with regular STRIPS operators. However, it should not be particularly surprising, because conditional operators are useful only when we have incomplete information about the initial state of the world, or the affects of the op-

erators, so that we can try to come up with a plan that would work in any situation that is consistent with the information available. Otherwise, we can replace the conditional operators with a number of ordinary STRIPS-style operators, to obtain an equivalent planning domain [10].

(ii) Comparing the complexity of PLAN EXISTENCE in the propositional case (in which all predicates are restricted to be 0-ary) with the datalog case (in which the predicates may have constants or variables as arguments) reveals a regular pattern. In most cases, the complexity in the datalog case is exactly one level harder than the complexity in the corresponding propositional case. We have EXPSPACE-complete versus PSPACE-complete, NEXPTIME-complete versus NP-complete, and finally EXPTIME-complete versus polynomial.

(iii) If delete lists are allowed, then PLAN EXISTENCE is EXPSPACE-complete but PLAN LENGTH is only NEXPTIME-complete. Normally, one would not expect PLAN LENGTH to be easier than PLAN EXISTENCE. In this case, it happens because the length of a plan can sometimes be doubly exponential in the length of the input. In PLAN LENGTH we are given a bound k, encoded in binary, which confines us to plans of length at most exponential in terms of the input. Hence finding the answer is easier in the worst case of PLAN LENGTH than in the worst case of PLAN EXISTENCE.

We do not observe the same anomaly in the propositional case, because the lengths of the plans are at most exponential in the length of the input. As a result, giving an exponential bound on the length of the plan does not reduce the complexity of PLAN LENGTH.

(iv) PLAN LENGTH has the same complexity regardless of whether or not negated preconditions are allowed. This is because what makes the problem hard is how to handle *enabling-condition interactions*. Enabling-condition interactions are discussed in more detail in [17], but the basic idea is that a sequence of actions that achieves one subgoal might also achieve other subgoals or make it easier to achieve them. Although such interactions will not affect PLAN EXISTENCE, they will affect PLAN LENGTH, because they make it possible to produce a shorter plan. It is not possible to detect and reason about these interactions if we plan for the subgoals independently; instead, we have to consider all possible operator choices and orderings, making PLAN LENGTH NP-hard.

(v) Delete lists are more powerful than negated preconditions. Thus, if the operators are allowed to have delete lists, then whether or not they have negated preconditions has no effect on the complexity.

5 Related Work

5.1 *Planning*

Bylander [3,4] has done several studies on the complexity of propositional planning. We have stated some of his results in Table 2. More recently, he has studied the complexity of propositional planning extended to allow a limited amount of inference in the domain theory [4]. His complexity results for this case range from polynomial time to PSPACE-complete.

Chapman was the first to study issues relating to the undecidability of planning; we have discussed his work in detail in Section 3.

Backstrom and Klein [2] found a class of planning problems called SAS-PUBS, for which planning can be done in polynomial time. Their planning formalism is somewhat different from ours: they make use of *state variables* that take values from a finite set, and consider a planning state to be an assignment of values to these state variables. Since they restrict each state variable to have a domain of exactly two values, we can consider each state variable to be a proposition; thus, in effect they are doing propositional planning. In order to get polynomial time results, they further restrict each operator to change at most one state variable, and do not allow more than one operator to change a state variable to a given value. When these restrictions are released, their planning algorithm performs in exponential time. It is not very easy to compare our results with theirs, because we use a different formalism, but we can safely state that we analyze a much broader range of problems.

Korf [18] has pointed out that given certain assumptions, one can reduce exponentially the time required to solve a conjoined-goal planning problem, provided that the individual goals are independent. Yang, Nau, and Hendler [35] have generalized this, showing that one can still exponentially reduce the time required for planning even if the goals are not independent, provided that only certain kinds of goal interactions are allowed. Under this same set of goal interactions, they have also developed some efficient algorithms for merging plans to achieve multiple goals [34,35].

Complexity results have been developed for blocks-world planning by Gupta and Nau [16,17] and also by Chenoweth [7]. Gupta and Nau [16,17] have shown that the complexity of blocks-world planning arises not from deleted-condition interactions as was previously thought, but instead from enabling-condition interactions. Their speculations that enabling-condition interactions are important for planning in general seem to be corroborated by some of our results, as discussed above.

Another problem that is closely related to planning is the problem of temporal projection, or what Chapman calls the "modal truth" of an atom [5]. Given an atom a, an initial state S_0, and a partially ordered set of actions P, the question is whether a is necessarily/possibly true after execution of P. This question is especially important in partial-order planners such as NOAH [28], NONLIN [32], and SIPE [33]. For example, McDermott [21] says "unfortunately, partial orders have a big problem, that there is no way of deciding what is true for sure before a step without considering all possible step sequences consistent with the current partial order," and Pednault [25] also expresses similar sentiments.

One problem is what it means for a to be necessarily true if not all total orderings of P are executable. Chapman [5] assumes that a is necessarily true after executing P only if every total ordering of P is both executable and achieves a; and in return, he comes up with a polynomial-time algorithm for determining the necessary truth of a. However, his algorithm does not work correctly for establishing the possible truth of a (Nau [23] proves that problem is NP-hard).

Chapman also proves that with conditional planning operators, establishing the necessary truth of a is co-NP-hard; and Dean and Boddy [8] prove a similar result with a more general notion of conditional planning operators (the same definition we gave in Section 2.2).[4] Dean and Boddy [8] also try to come up with approximate solutions for the problem. They present algorithms for computing a subset of the propositions that are necessarily true, and for computing a superset of the propositions that are possibly true. Furthermore, the complexity of these algorithms is polynomial if the number of triples for each operator is bounded with a constant. However, we do not know of any results concerning how close the approximations are.

6   Conclusions and Future Work

Although our equivalence between planning and logic programming only holds in certain limited cases, this equivalence has allowed us to transport many decidability and undecidability results from logic programming to planning. Among other things, our results solve an open problem posed by Chapman [5], and clear up some difficulties with his undecidability theorems. It is not a

---

[4] In both cases, they state that the problem is NP-hard, but their proofs establish co-NP-hardness.

trivial task to extend this equivalence, because negation has different semantics for logic programming and planning. One recent result in this direction is the following: Subrahmanian and Zaniolo [30] have shown that STRIPS-style planning (with delete lists and negative preconditions) can be transformed, in polynomial time, to a class of logic programs with negation. Based on this transformation, they show how logic programming update techniques can be used to handle "surprises" that may occur during plan execution.

For those cases where planning is decidable, we have shown how the time complexity varies depending on a wide variety of conditions. Our results suggest that for finding optimum plans, enabling-condition interactions (first described by Gupta and Nau [17]) can be just as important as the better-known deleted-condition interactions. In addition, we have also shown that delete lists are more powerful than negated preconditions, and that conditional operators do not affect the complexity of planning. Thus, negated preconditions and conditional operators can be incorporated into planning systems, improving flexibility and usability without much cost.

Although the past few years have seen much analysis of the properties of total- and partial-order planning systems using STRIPS-style planning operators [22,20,5], a more popular approach for practical work on AI planning systems is hierarchical task-network (HTN) decomposition [28,32,33]. However, there has been very little analytical work on the properties of HTN planners. One of the primary obstacles impeding such work has been the lack of a clear theoretical framework explaining what a HTN planning system is. To address this problem, some of us (together with Jim Hendler) are developing a formalization of HTN planning [12,9]. We intend to use this formalism to correctly define, explicate, and analyze various properties of HTN planning systems, such as soundness, completeness, complexity, and expressivity.

Acknowledgement

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1976.

[2] Christer Backstrom and Inger Klein. Planning in polynomial time: the sas-pubs class. *Computational Intelligence*, 7, 1991.

[3] Tom Bylander. Complexity results for planning. In *IJCAI-91*, 1991.

[4] Tom Bylander. Complexity results for extended planning. In *Proc. First International Conference on AI Planning Systems*, 1992.

[5] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–379, 1987.

[6] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, MA, 1985.

[7] Stephen V. Chenoweth. On the NP-hardness of blocks world. In *AAAI-91: Proc. Ninth National Conf. Artificial Intelligence*, pages 623–628, July 1991.

[8] Thomas Dean and Mark Boddy. Reasoning about partially ordered events. *Artificial Intelligence*, 36:375–399, 1988.

[9] K. Erol, J. Hendler, and D. Nau. Complexity results for hierarchical task-network planning. 1993. Submitted for journal publication.

[10] K. Erol, D. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. Technical Report CS-TR-2797, UMIACS-TR-91-154, SRC-TR-91-96, Computer Science Department and Institute for Systems Research, University of Maryland, College Park, MD, 1991.

[11] K. Erol, D. Nau, and V. S. Subrahmanian. When is planning decidable? In *Proc. First Internat. Conf. AI Planning Systems*, pages 222–227, June 1992.

[12] K. Erol, D. S. Nau, and J. Hendler. Toward a general framework for hierarchical task-network planning. In *AAAI Spring Symposium*, April 1993.

[13] 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.

[14] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[15] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: a Foundation for Computer Science*. Addison-Wesley, 1989.

[16] Naresh Gupta and Dana S. Nau. Complexity results for blocks-world planning. In *Proc. AAAI-91*, 1991. Honorable mention for the best paper award.

[17] Naresh Gupta and Dana S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3):223–254, August 1992.

[18] Richard Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, September 1987.

[19] Vladimir Lifschitz. On the semantics of STRIPS. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 523–530. Morgan Kaufman, 1990.

[20] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *AAAI-91*, pages 634–639, July 1991.

[21] Drew McDermott. Regression planning. *International Journal of Intelligent Systems*, 6:357–416, 1991.

[22] S. Minton, J. Bresna, and M. Drummond. Commitment strategies in planning. In *Proc. IJCAI-91*, July 1991.

[23] D. Nau. On the complexity of possible truth. In *AAAI Spring Symposium*, April 1993.

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

[25] Edwin Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4:356–372, 1988.

[26] M. A. Peot. Conditional nonlinear planning. In *Proc. First International Conference on AI Planning Systems*, pages 189–197, 1992.

[27] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[28] Earl D. Sacerdoti. The nonlinear nature of plans. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 162–170. Morgan Kaufman, 1990. Originally appeared in *Proc. IJCAI-75*, pp. 206-214.

[29] J. Shoenfield. *Mathematical Logic*. Academic Press, 1967.

[30] V.S. Subrahmanian, C. Zaniolo. Database Updates and AI Planning Domains. Technical Report CS-TR-3173, UMIACS-TR-93-118, Computer Science Department, University of Maryland, College Park, MD, 1993.

[31] A. Tate, J. Hendler, and M. Drummond. A review of ai planning techniques. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 26–49. Morgan Kaufman, 1990.

[32] Austin Tate. Generating project networks. In *Proc. 5th International Joint Conf. Artificial Intelligence*, 1977.

[33] David E. Wilkins. Domain-independent planning: Representation and plan generation. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 319–335. Morgan Kaufman, 1990. Originally appeared in *Artificial Intelligence* 22(3), April 1984.

[34] Q. Yang, D. S. Nau, and J. Hendler. Optimization of multiple-goal plans with limited interaction. In *Proc. DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, 1990.

[35] Q. Yang, D. S. Nau, and J. Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8(2):648–676, February 1992.