Journal of Artificial Intelligence Research 14 (2001) 1-28 Submitted 2/00; published 1/01

On Reachability, Relevance, and Resolution in the Planning as Satisfiability Approach

Ronen I. Brafman BRAFMAN@CS.BGU.AC.IL
Department of Computer Science, Ben-Gurion University
P.O.Box 653, Beer-Sheva 84105, Israel

Abstract

In recent years, there is a growing awareness of the importance of reachability and relevance-based pruning techniques for planning, but little work specifically targets these techniques. In this paper, we compare the ability of two classes of algorithms to propagate and discover reachability and relevance constraints in classical planning problems. The first class of algorithms operates on SAT encoded planning problems obtained using the linear and GRAPHPLAN encoding schemes. It applies unit-propagation and more general resolution steps (involving larger clauses) to these plan encodings. The second class operates at the plan level and contains two families of pruning algorithms: Reachable-k and Relevant-k. Reachable-k provides a coherent description of a number of existing forward pruning techniques used in numerous algorithms, while Relevant-k captures different grades of backward pruning. Our results shed light on the ability of different plan-encoding schemes to propagate information forward and backward and on the relative merit of plan-level and SAT-level pruning methods.

1. Introduction

The success of the planning as satisfiability (PAS) approach (Kautz & Selman, 1992, 1996) has led to various attempts to refine the initial methods used and to improve our understanding of its performance. In particular, various methods for generating formulas from planning instances have been compared (Ernst, Millstein, & Weld, 1997), and various systematic alternatives to the original stochastic method have been examined (e.g., Bayardo & Schrag, 1997; Li & Anbulagan, 1997). Still, many issues surrounding this approach are poorly understood. In particular, little is known about the influence of the encoding method on performance.

Concentrating on the two encoding methods proposed by Kautz and Selman (1996), the linear and the GRAPHPLAN-based encodings, we examine their influence on the ability to propagate reachability and relevance information via unit propagation and, more generally, k-clause resolution. We do so by comparing the pruning ability of these techniques to that of a class of algorithms for reachability and relevance analysis that operate on the original problem formulation: Reachable-k and Relevant-k. Reachable-k is a simplified variant of a similar algorithm for state pruning in Markov decision processes (Boutilier, Brafman, & Geib, 1998), while Relevant-k is a natural counterpart used for relevance analysis. Both algorithms provide a coherent framework for discussing different grades of reachability and relevance-based pruning methods that appear in the literature.

Our work is motivated by the growing role that forward and backward pruning methods play in current planning algorithms and the important role of propagation techniques in all SAT solvers used in the planning as satisfiability approach. Unit propagation plays a central role in the Davis-Putnam algorithm (Davis & Putman, 1960) and many of its offsprings (e.g., Crawford & Auton, 1993; Freeman, 1995; Gomes, Selman, & Kautz, 1998; Li & Anbulagan, 1997) and it is used as a preprocessing step when stochastic methods are applied. Moreover, a limited form of binary propagation is used in Crawford's Compact program for simplifying CNF formulas which is utilized in the Blackbox planner (Kautz & Selman, 1999). Our results shed some light on the relationship between these pruning techniques.

The paper is organized as follows. Section 2 provides background material, describing the basic ideas of the PAS framework and the Graphplan algorithm. In Section 3, we discuss Reachable-k, an algorithm for performing reachability analysis, and compare its ability to prune possible actions to that of k-clause resolution. In Section 4 we describe Relevant-k which is similar to Reachable-k and is applied to relevance analysis. Again, we compare it to methods based on resolution. In Section 5 we empirically compare the results of various methods for $k = 1, 2$. We conclude with a discussion of future and related work in Section 6. Proofs appear in the appendix, but their main arguments are described in the body of the paper.

2. Background

The Graphplan algorithm (Blum & Furst, 1997) and the Satplan algorithm (Kautz & Selman, 1996) have profoundly altered the direction of research within the planning community. Two of the main concepts studied in this paper, reachability analysis and plan encodings, have become central to current planning research thanks to these planners. We briefly discuss these planners, and in particular, their aspects pertaining to our topic.

2.1 Reachability Analysis in Graphplan

The purpose of reachability analysis is to discover unreachable states of the world and infeasible actions, i.e., actions that cannot be performed in the course of a successful plan. By discovering such actions ahead of time, we reduce the space that needs to be searched to find a valid plan. In principle, full fledged reachability analysis requires forward search in the space of possible states. This is a very expensive operation, and instead, we can opt for sound, but incomplete methods. Such methods do not discover all the actions that can be ruled out. However, any action that they rule out is infeasible and need not be considered when searching for a plan.

The Graphplan planner provides a good example of the utility of approximate reachability analysis. Graphplan has two main stages: In the first stage, approximate reachability analysis is conducted, yielding a data-structure called the *planning graph* which represents a sound, but incomplete, approximation of the set of states reachable from the initial state. In the second stage, Graphplan searches for a plan over this structure. Graphplan's planning graph construction algorithm presents a particularly good tradeoff between computational complexity and pruning power, and its utility in pruning the search space is attested to by the planner's good performance.

The planning graph construction algorithm can be viewed as generating a list of annotated sets. The odd elements in this list contain sets of propositions. The even elements in this list contain sets of actions. Each such set is annotated with mutual exclusion constraints on its members. Intuitively, the i^{th} action set contains the list of actions that could be performed at the i^{th} step of a concurrent action plan (i.e., a plan allowing for concurrent execution of actions that do not interfere with each other). The i^{th} proposition set contains propositions that could hold after $i-1$ (sets of concurrent) actions have been performed. The mutual exclusion constraints circumscribe these sets by indicating that certain pairs of actions or propositions cannot occur at the same time at a particular stage. Hence, if the propositions p, q appear in the i^{th} proposition set, then it is possible (or more accurately – the algorithm does not rule out the possibility) that an i-step plan applied at the initial state could lead to a world in which p and/or q hold. However, if p and q are marked mutually exclusive then we know that p and q cannot hold *together* after some i-step plan has been performed.

The sets are constructed as follows: The first proposition set contains the propositions that hold at the initial state. The first action set contains the actions that can be performed at the initial state. In general, the i^{th} set of propositions contains the effects of actions in the i-1^{th} action set, and the i^{th} action set contains actions whose preconditions appear in the i-1^{th} set of propositions, provided that their preconditions are not marked as mutually exclusive. Mutual exclusion constraints are added as follows: Two actions are marked mutually exclusive at the i^{th} action set if their preconditions are marked mutually exclusive at the i-1^{th}. Clearly, if all the preconditions of these actions cannot hold at this time point, we cannot perform both actions together at this time point. Another reason for marking actions as mutually exclusive is if they conflict. That is, if one action destroys a precondition or an effect of the other action. Two propositions at the i^{th} proposition set are marked mutually exclusive if all pairs of actions in the i-1^{th} action set that have them as effects are mutually exclusive. When that is the case, there is no way for us to achieve both of these propositions together at this stage.

The initial construction of the planning graph is terminated once all goal propositions appear in the last proposition set. At that point, Graphplan performs a form of regression-based search on the planning-graph structure. If this search fails, the planning graph is extended by one additional layer of actions and one additional layer of propositions and searched again. For more details, see the article by Blum and Furst (1997).

2.2 The Planning as Satisfiability Approach

The Planning as Satisfiability approach (PAS for short), works as follows: given a planning problem and a bound n on the size of the plan, the *plan encoder* generates a propositional formula in conjunctive normal form. This formula has the following property: it is satisfiable iff the planning problem has a solution with at most n time steps. Intuitively, the formula is composed of propositions describing the state of the world throughout the execution of an n-step plan.

The propositional language on top of which the formula is defined contains a proposition for each possible aspect of the world at each time point. For example, suppose we are looking at the blocks' world domain where states are described using the ON and CLEAR relations. In that case, for any pair of blocks x, y, and any time point $0 \leq t \leq n$ we shall introduce a proposition $p_{\mathrm{ON}(x,y,t)}$ which corresponds to x being on top of y at time t. Similarly, for every block x and time point $0 \leq t \leq n$, we shall introduce a proposition $p_{\textsc{clear}(x,t)}$ which corresponds to x being clear at time t.

A truth assignment to the language described above can be viewed as describing the state of the world during the execution of an n-step plan. For instance, if $p_{\textsc{on}(A,B,3)}$ is assigned *true*, then block A is on top of block B at time 3. Of course, most truth assignments to this language would *not* correspond to anything resembling the true state of the world during the execution of an actual plan. For example, both ON(A,B) and ON(B,A) could be assigned *true*. The goal of the encoding scheme is to generate a formula such that any assignment satisfying this formula will correspond to the true state of the world during the execution of an actual plan that achieves the desired goal. Each axiom in this formula places some constraint on the value of these propositions so that the combined effect of these constraints is to ensure that the resulting formula has precisely the desired truth assignments. For example, one type of axiom will be a conjunction of all the propositions corresponding to the initial state. Any truth assignment satisfying this axiom must ensure that these propositions hold at time 0. Another class of axioms could state that if some action a is performed at time t then the world at time $t-1$ must satisfy the preconditions of a. In the next sections we discuss two of the central plan encodings in more detail.

Once an appropriate formula has been generated, it is simplified using various well known techniques. In particular, all simplifiers employ a unit-resolution step (Genesereth & Nilsson, 1987). Unit resolution (also known as *unit propagation*) works as follows: To satisfy a CNF formula, we must satisfy each of its clauses. In particular, if one of the clauses contains a single literal (such a clause is known as a *unit clause*) we immediately know that the variable appearing in this clause must be assigned an appropriate value. Any clause containing the same literal will be satisfied now, and it can be removed from the formula. Any clause c containing the negation of that literal can be resolved against this unit clause, and the resulting clause (which is smaller than c) can replace c. For example, suppose we have the formula $(p) \wedge (\neg q \vee \neg p) \wedge (r \vee p)$. The first disjunct, (p) is a unit clause. Hence, p must be assigned *true*. This makes the third clause, $(r \vee p)$, satisfied. The second clause is now resolved with the first clause, and we replace $(\neg q \vee \neg p)$ by $(\neg q)$. We now have a new unit clause, $(\neg q)$, and so the proposition q must be assigned the value *false*. If we had additional clauses containing q or $\neg q$, we could remove them, or simplify them, as appropriate.

After simplification, we apply our favorite algorithm for finding satisfying assignments, and if one is attained, a *decoder* is used to obtain an actual plan from this solution. If we do not find a satisfying assignment, we increase the value of n (the size of the plan), and try again.

Finally, we note that the BLACKBOX planner (Kautz & Selman, 1999) combines PAS with GRAPHPLAN's reachability analysis. It constructs a planning graph, and uses it to generate a particular plan encoding.

3. Reachability and Resolution

Reachability and relevance analysis form an essential part of successful modern planning algorithms. The most notable example of reachability analysis is GRAPHPLAN's planning graph (Blum & Furst, 1997), and many recent planners employ either reachability analysis (e.g., Bonet, Loerincs, & Geffner, 1997), relevance analysis (e.g., McDermoot, 1996; Nebel, Dimopoulos, & Koehler, 1997), or both (Kambhampati, Parker, & Lambrecht, 1997). The importance of reachability and relevance analysis has been noted in the context of decision-theoretic planning as well. For example, Boutilier and Dearden (1994) employ relevance analysis to reduce the state-space, and Boutilier, Brafman, and Geib (1998) describe a general method for reachability analysis for MDPs. Below, we discuss this method in a simplified form suitable for classical planning problems described using the Strips representation language (Fikes & Nilsson, 1971). In Section 4, we present a counterpart of this method for performing relevance analysis and relate these algorithms to k-clause resolution in the context of SAT-encoded planning problems.

3.1 Propagating Reachability Information

Reachable-k (Boutilier, Brafman, & Geib, 1998) is an algorithm for estimating the states reachable from a given initial state. As formulated, it is quite general and applies to domains with non-deterministic actions and conditional effects. In Figure 1, we present a simplified version of that algorithm, Reachable-k, which deals with deterministic, unconditional actions represented in the Strips representation language. A prolog implementation of Reachable-k is available at www.cs.bgu.ac.il/~pdm.

An important reason for our interest in Reachable-k is its similarity to the influential planning graph construction of the Graphplan planner (Blum & Furst, 1997). In fact, it generalizes the ideas behind Graphplan's planning graph, which is equivalent to Reachable-2. We use A_i to denote the set of actions feasible i steps from the initial state, S_i to denote the corresponding set of propositions, and CS_i^* to denote constraints on these propositions, such that if $\{p_1, \ldots, p_m\} \in CS_i^*$ then these propositions cannot co-occur after i steps. CA_i^* denotes similar constraints on actions. Here, $* \in \{L, P\}$, where L is used when we restrict our attention to linear action sequences, and P is used when we allow concurrent non-conflicting actions (i.e., actions that do not destroy each others' effects or preconditions and whose preconditions are not constrained not to co-occur). Of course, for $k = 1$ the sets CS_i^* and CA_i^* are empty for all $i \geq 0$. (Actually, as defined, CA_i^L is not empty even when $k = 1$, but it plays no real part in the algorithm, and it should be ignored). Finally, note that in this description, the set of possible actions contains all actions of the form NOOP[l], where l is a literal.

When $k = 2$, S_i and A_i represent the propositional and action levels of Graphplan's planning graph, and CS_i and CA_i^P hold their respective mutual exclusion constraints.

We have not stated a termination condition for Reachable-k, but one can be formulated based on the content of S_i or the index i itself. In the PAS framework, where the number of time-steps is fixed, one would opt for the second alternative. Reachable-k gives us sets of actions and propositions, A_j, S_j, that could occur after the performance of j actions (or j sets of concurrently non-conflicting actions) from the initial state. It is easy to see that Reachable-k is sound in the following sense:

Theorem 1 *If a set of propositions or actions is excluded by Reachable-k at time j then there is no feasible plan in which, at time j, these propositions hold or, respectively, these actions appear.*

- $S_0 =$ literals that hold in the initial state.
- $CS_0 = \{\}$.
- $A_0 =$ actions all of whose preconditions are in S_0.
- $CA_0^L = \{\{a_i, a_j\} | a_i, a_j \in A_0,\ i \neq j,$ neither a_i nor a_j are noops or a_j is a noop whose effect is destroyed by $a_i\}$.
- $CA_0^P = \{\{a_i, a_j\} | a_i, a\, a.$

the Reachable-k algorithm. By reachability information we mean constraints on the set of actions possible at a time point or constraints on world states (in the form of, e.g., sets of unreachable propositions or k-tuples of propositions). Hence, for example, a constraint of the form $a(t) \vee a'(t)$ implies that one of the actions a or a' must appear at time t in the plan. A constraint of the form $\neg a(t) \vee \neg a'(t)$ implies that one of the actions a or a' must not appear in the plan. Similar constraints on the propositions holding at a time point can also be derived. In principle, such constraints reduce our search space and could help us attain a solution more quickly. However, the effectiveness of such deduced constraints depends on the precise algorithm used. Moreover, comparison over a very large class of constraints seems quite difficult. Therefore, in this article we concentrate on a very concrete class of reachability information of the form $\neg a(t)$, i.e., the action a cannot be performed at any state reachable via t steps. These are powerful constraint which can be utilized effectively by almost all planners (perhaps with the exception of partial-order planners). Consequently, we shall say that an algorithm Alg_1 generates *more* reachability information than another algorithm Alg_2 if whenever Alg_2 is able to determine that some action a cannot be performed at some time t, Alg_1 is able to reach this conclusion as well, and in addition, there are such conclusions which Alg_1 can reach but which Alg_2 cannot reach. Hence, Alg_1 generates a strict superset of the constraints on actions (of the type we are interested in) generated by the other algorithm. Note that this does not mean that Alg_1 is better than Alg_2 on every instance, only that it is always as good, and in some cases better. In this section we shall compare the pruning ability of the two Reachable-k variants and two encoding methods discussed by Kautz and Selman (1996):[2] the linear encoding and the GRAPHPLAN encoding.

3.2.1 LINEAR PLAN ENCODING

The linear plan encoding (Kautz & Selman, 1992) is a simple and natural method for translating a planning problem into a formula that is satisfiable iff there is a valid plan of length n (for some given n). The clauses in the linear plan encoding fall into the following classes:

1. an action implies its preconditions prior to its execution;
2. an action implies its effects following its execution;
3. an action does not affect any other proposition (frame axioms);
4. there is at least one action at each time point;
5. there is at most one action at each time point.

Because we have explicit frame axioms, noops are not needed in the linear encoding (as opposed to the GRAPHPLAN encoding). In addition, the formula contains unary clauses describing the initial and goal states. However, for the purpose of analyzing reachability effects, we *exclude* the description of the goal state (which plays a role in relevance analysis).

Consider the mechanism by which resolution can yield reachability information: Given the propositions that hold at the initial state, we can derive the negation of actions whose

---

[2] The third (state-based) encoding method cannot be generated automatically.

preconditions do not hold using unit propagation on axioms of class 1. Propagating these unit clauses with the appropriate instance of axiom class 4, we will obtain a disjunction of all actions that can be executed at the first time point. So far, this is identical to what Reachable*-k provides. To propagate this information forward, we can resolve these action disjunctions with axioms of class 2 and 3. This, however, requires binary resolution (discussed below). Hence, except for the unlikely case in which a single action is possible, there is no more that we can derive using unit propagation alone. Reachable*-1, on the other hand, can provide us with a list of all possible effects of these actions and possibly prune out future actions whose preconditions do not appear in this list. We conclude:

Lemma 1 *In the context of the linear encoding, Reachable*-1 *yields more reachability information than unit propagation.*

Example: Consider a blocks' world domain with a single action schema MOVE(object,source, destination).[3] Its preconditions are: ON(object,source), CLEAR(object), CLEAR(destination) and its effects are: ON(object,destination), CLEAR(source), ¬ON(object,source), ¬ CLEAR (destination) (except when the destination is the table which is always clear). If we have k stacks of blocks initially, k^2 actions can be performed at the initial state (i.e., moving a block from the top of a stack to the top of another stack or the table). This will be discovered by both algorithms. In particular, unit propagation will yield a disjunction of all these actions. We know that all blocks that are 2 or more blocks below the top cannot participate in the second MOVE action. Reachable-1 will find this out due to the fact that they are not CLEAR. Suppose that A is one such block. All initially feasible MOVE actions participate in a frame axiom of the form MOVE(o,s,d)∧¬CLEAR$(A,0)$ → ¬CLEAR$(A,1)$, which, in clausal form is ¬MOVE(o,s,d)∨CLEAR$(A,0)$ ∨ ¬CLEAR$(A,1)$. Resolving against ¬CLEAR$(A,0)$, we have ¬MOVE(o,s,d)∨¬CLEAR$(A,1)$. If we could deduce ¬CLEAR$(A,1)$, we could rule out all actions that have it as a precondition. But if we are restricted to unit propagation, this requires deducing MOVE(o,s,d) for some initially feasible action, and we cannot make such a deduction. ∎

If we propagated information forward using axioms of class 2 and 3 and used binary resolution (as discussed before Lemma 1), we now have a set of disjunctions of the possible effects (including frame effects) of the initially allowable actions. The number of such disjuncts is $O(e^m)$, where e is the maximal number of effects of an action and m is the number of actions that can be executed initially. In some cases, these disjunctions could contain a single literal, e.g., when all initially allowable actions leave some proposition unchanged. When one of these disjunctions contains only literals that are negations of some action's precondition, we can deduce the negation of this action by resolving with axioms of class 1.

Example: In the example considered above we would generate a disjunction of the form MOVE(o_1, s_1, d_1) ∨ MOVE(o_2, s_2, d_2) ∨ MOVE(o_3, s_3, d_3), containing all instances of the MOVE action for time 0 whose negations have not been deduced. As discussed above, for all such actions, we can obtain a clause of the form ¬MOVE(o_i, s_i, d_i)∨¬CLEAR$(A,1)$. Once we resolve these binary clauses against the clause above, we obtain a unary clause ¬CLEAR$(A,1)$, that

---

[3] In fact, since we use plain STRIPS, we need three action schemas: one for moving a block to a block, one for moving a block to a table, and one from moving a block from the table. However, as this does not affect our analysis, we stick to a single MOVE action in this and the following examples.

can be used in conjunction with class 1 axioms to deduce the negations of step 2 actions whose preconditions include $\text{CLEAR}(A, 1)$. ∎

As we saw, the effect disjunctions discussed above allow us to rule out certain propositions or combinations of propositions. These are analogous to mutual exclusion constraints. These mutual exclusion constraints can be used to prune actions. For example, if we deduce $\neg p_1 \vee \cdots \vee \neg p_m$ and all the p_i are preconditions of some action a, we can deduce $\neg a$ using binary resolution (by resolving precondition axioms with this disjunction). However, as we show below, binary resolution has trouble propagating even binary mutual exclusion constraints forward. We believe that this is generally true, i.e., k-clause resolution will have trouble propagating k-ary constraints. We can show the following:

Lemma 2 *Reachable-2 and binary resolution (in the case of the linear encoding) are incomparable.*

We prove this by providing two examples. One in which Reachable-2 is able to prune an action that binary resolution cannot, and one in which the converse hold.

First, consider the 4-bit counter with initial value 0000 (i.e., $\neg p_1, \neg p_2, \neg p_3, \neg p_4$). After four steps we obtain the following: $S_4 = \{\neg p_1, p_1, \neg p_2, p_2, \neg p_3, p_3, \neg p_4\}$ and $CS_4 = \{(p_1, p_3), (p_2, p_3)\}$. Therefore, $A_4 = \{a_1, a_2, a_3\}$. This implies that $S_5 = S_4$. We claim that $(p_2, p_3) \in CS_5$ as well, which means that $a_4 \notin A_5$. To see this, consider all pairs of actions that have p_2 and p_3 as effects. They are: (a_2, a_3), $(a_2, \text{NOOP}[\text{P}_3])$, $(\text{NOOP}[\text{P}_2], a_3)$, and $(\text{NOOP}[\text{P}_2], \text{NOOP}[\text{P}_3])$. (a_2, a_3) is a pair of real actions, which are always mutually exclusive in the linear encoding. The preconditions of $(a_2, \text{NOOP}[\text{P}_3])$ are mutually exclusive according to CS_4, and so are the preconditions of $(\text{NOOP}[\text{P}_2], \text{NOOP}[\text{P}_3])$. Finally, $(\text{NOOP}[\text{P}_2], a_3)$ are interfering actions. We conclude that $(p_2, p_3) \in CS_5$ and $a_4 \notin A_5$.

When we run a binary resolution procedure on the linear encoding of this problem, we could not deduce $a_4 \notin A_5$. This stems from the fact that ternary resolution is needed to propagate the mutual exclusion of p_2 and p_3. Recall that we obtain mutual exclusion constraints by resolving against a disjunction of actions that have not been ruled out. In the above case, at time 4 we would have the following disjunction: $a_1^4 \vee a_2^4 \vee a_3^4 \vee \text{NOOP}[\neg \text{P}_1] \vee \cdots \vee \text{NOOP}[\neq \text{P}_4]$. Our goal is to deduce $\neg p_2^5 \vee \neg p_3^5$ using $\neg p_2^4 \vee \neg p_3^4$ and the various axioms. To do this, we will try to deduce either $\neg p_2^5 \vee \neg p_3^5$ from each of the actions in the disjunction. It is easy to deduce $\neg p_2^5$ from a_3^4 and $\neg p_2^5$ from a_2^4. However, we believe that it is impossible to deduce $\neg p_2^5 \vee \neg p_3^5$ from a_1^4 and from some of the noops.[4] The reason for this is that such a deduction involves the use of frame axioms, which are ternary. If we know that, e.g., $\neg p_2^4$ holds, we apply unit resolution to the frame axioms and obtain a binary clause. However, here we only know $\neg p_2^4 \vee \neg p_3^4$. Once we resolve this against a frame axiom we remain with a ternary clause. To get our desired result we must resolve two such ternary clauses.

Finally, let us see an example in which we use binary resolution to derive a ternary constraint. By definition, Reachable-2 cannot derive such constraints. Suppose that the initial state is $\neg p, \neg q, \neg r$. We have four actions: $a1$ has p, r as effects, $a2$ has q, r as effects, $a3$ has p, q as effects, and $a4$ has p, q, r as preconditions. Using Reachable*-2 we deduce that $a1, a2, a3$ are possible at time 0. We get as their possible effects $p, q, r, \neg p, \neg q, \neg r$ (recall

---

[4] The fact that the deduction is impossible has been verified. What we are hypothesizing here is the reason for it.

that we must include all noop actions in Reachable-k in order to capture frame effects). No strict subset of p, q, r can appear in the set of constraints CS_1^*. Since we deal with binary constraints only, the set $\{p, q, r\}$ does not appear in CS_1^*. Therefore, we will consider $a4$ possible at time 1, although, in fact, it is impossible. Using binary resolution, we would have obtained the constraint $\neg p \vee \neg q \vee \neg r$ (referring to time 1) which would have enabled us to deduce that a_4 is impossible at time 1. ∎

3.2.2 The Graphplan Encoding

The Graphplan encoding differs from the linear encoding by its ability to consider multiple concurrent (non-interfering) actions, allowing one to obtain shorter plans which, in turn, can reduce the search space size. It constructs the following sets of clauses:

1. An action implies its preconditions;
2. An effect implies one of the actions that has this effect;
3. There is at least one action at each time-point;
4. Two conflicting actions cannot occur together.

Besides the obvious ability to consider multiple parallel (non-interfering) actions, the important difference between the Graphplan and Linear encoding is in axiom class 2 (referred to in (Ernst et al., 1997) as *explanatory* frame axioms.) Clauses in this class will contain positive occurrences of action literals and negative occurrences of state literals.

As in the linear case, using unit propagation we can infer which actions cannot be applied at the initial state. Using axioms of class 2, we can propagate this information forward, deducing the negation of all effects that cannot be produced by the initially allowable actions. This information enables us to exclude actions whose preconditions cannot be produced. This forward propagation is essentially identical to Reachable-1. We can informally conclude:

Lemma 3 *In the context of the* Graphplan *encoding, unit propagation and Reachable-1 yield the same reachability information, if we ignore the explicit constraints appearing in axiom class 4. If we use these constraints, unit propagation can yield more reachability information.*

To be precise we have to carefully define the notion of reachability constraints in the context of the Graphplan encoding. For example, in the Graphplan encoding we can derive a constraint that says that one of a group of actions must appear in the plan. This constraint will not necessarily rule out any action because the Graphplan encoding permits multiple actions at the same time point.[5] However, in the linear encoding such a constraint will immediately rule out all other actions because only a single action is allowed at each time point. As we mentioned earlier, in this paper we concentrate on strict exclusion constraints

---

[5] However, because actions that interfere with each other cannot occur concurrently, if we know that action a will occur then we can deduce that any action a' that interferes with a will not occur. This is precisely where class 4 axioms enter the picture.

which lead to an immediate reduction in the search space by ruling out the need for certain actions at certain time points.

When $k > 1$, the mechanism remains the same. But now, axioms of class 4 can play a more prominent role because we can use them to exclude actions in more cases than before. However, the same problem of propagating mutual exclusion constraints forward which we had with the linear encoding reappear here. Consequently, k-clause resolution in the context of the Graphplan encoding and Reachable-k are incomparable.

4. Relevance and Resolution

Relevance analysis is a complex task and it can be performed to various degrees. For instance, considering the last action level, one can exclude actions that do not produce a literal in the goal. However, some actions producing a goal literal can also be irrelevant. For example, consider a blocks' world planning problem in which the color of the blocks is specified as part of the goal. As observed by Nebel, Dimopoulos, and Koehler (1997), a *paint-block* action is still, intuitively, irrelevant if the initial and final colors of the blocks are the same. However, it does have a goal literal as an effect.

In this section, we formulate an algorithm for relevance analysis, called Relevant-k. Relevant-k does not perform the deeper relevance analysis needed to determine that the *paint-block* action is irrelevant in the above example. Rather, Relevant-k is similar in its motivation and form to Reachable-k, and it has a similar soundness property. Relevant-k prunes the search space by excluding states from which the goal is not reachable within a given number of steps and actions that are not useful for achieving the goal state within a given number of steps.

Relevant-1 is similar to a number of existing components of existing planners, such as McDermott's *greedy regression graph* (McDermoot, 1996) and Nebel, Dimopoulos, and Koehler's And-Or trees (Nebel et al., 1997). Relevant-k generalizes these ideas to arbitrary levels of interactions, taking into consideration mutual-exclusion constraints that relevant states must satisfy. Relevant-k is slightly more complicated then Reachable-k because the Strips formalism allows incomplete description of goal states, and propagating this partial information raises some difficulties. Naturally, if the goal state is partially specified, fewer constraints are available to start with, and so fewer constraints will be derived. The algorithm is described in Figure 2. We are not aware of a similar, general formulation of these ideas. Therefore, it is worthwhile going over the central points of this algorithm, concentrating on the more interesting and complex case in which parallel actions are allowed. However, before we do this, we point out an important assumption we shall make on the action representation used: No proposition symbol shall appear only in the preconditions or only in the effects of an action. This restriction is not difficult to enforce, as any Strips-based domain representation can be transformed into a description in which this assumptions is satisfied. For example, if p is a precondition of action a that does not appear in the effect of a, we can simply add it to the effect, as we know that it must hold after the action is executed. If p appears in the effect of a but neither p nor $\neg p$ appear in the preconditions of a, we can decompose a into two versions of the a action, one in which p is a precondition and one in which $\neg p$ is a precondition. Note that in the worst case, such a transformation can cause an exponential blow-up in the number of actions.

- A_0^r contains all actions that are useful and safe w.r.t. the goal.
- A_0 contains A_0^r and all noops that are safe w.r.t. the goal.
- CA_0 contains all pairs of interfering actions in A_0.

We define R_i, S_i, A_i^r, A_i inductively as follows:

- R_i is the union of preconditions of actions in A_{i-1}^r.
- S_i is the union of preconditions of actions in A_{i-1}.
- CS_i contains sets S of literals such that $S \subset S_i$, $|S| \leq k$ and for any set of actions $A \subset A_{i-1}$ whose preconditions contain S it is the case that $A \in CA_{i-1}$.
- A_i^r contains all actions that are useful w.r.t. R_i but no subset of their effects is contained in CS_i.
- A_i contains A_i^r and all noops useful w.r.t. S_i.
- CA_i contains all action sets A such that $A \subset A_i$ and either (1) A contains two interfering actions, or (2) Some subset of the set of effects of A is in CS_i.

Action descriptions must contain the same set of propositional symbols in their precondition and effect lists.

Figure 2: The Relevant-k Algorithm

For $k = 1$ the algorithm is quite simple (and identical in the parallel and linear cases). In that case, we can ignore the sets S_i, CS_i, A_i and CA_i (as they are degenerate) and consider the sets R_i and A_i^r only. Starting with the goal literals, at each stage we have a set of literals from which we construct the next set of actions. This action set contains actions with an effect in the current literal set. However, if all the goal effects of an action are all part of its preconditions, we can ignore that action as irrelevant. Next, a new literal set is constructed, containing the set of preconditions of the current set of actions, and we repeat the process with this new set.

When $k > 1$, the picture becomes a bit more complicated. We start with the set of relevant actions, A_i^r. These are actions that achieve one of the desired literals. In particular, A_0^r contains only actions that have one of the goal literals as an effect (but not as a precondition). If the goal is partially specified, literals that are not part of it could hold in the previous time step. Hence, we include the appropriate noop actions in a larger set, A_i, which contains both A_i^r and noops that do not destroy needed propositions. A subset of the actions in A_i is mutually exclusive if it contains interfering actions or actions whose effects are mutually exclusive. Given the set A_{i-1}^r, we generate the set R_i, which includes the preconditions of A_{i-1}^r. The set S_i is defined as the set of preconditions of actions in A_i. If the goal is a completely specified state, there is no the sets R_i and S_i and the sets A_i^r and A_i are identical, and so we need not distinguish between them.

To facilitate the description of the Relevant-k algorithm, it would be useful to add a few simple definitions. First, we wish to revise the definition of *interfering actions* in the context of the Relevant-k algorithm. We say that actions a, a' *interfere* with each other if some effect of a conflicts with some precondition or effect of a' or (and this is beyond the previous definition of this term) if their preconditions are inconsistent. An action a is *useful* w.r.t. (with respect to) some literal l if a is the noop action preserving l or l is an effect, but not a precondition, of a. a is *useful* w.r.t. some set of literals if it is useful w.r.t. one of the set's elements. A set A of actions is *safe* w.r.t. some set of S of literals if no action in A has an effect that negates an element of S.

Relevant-k embodies the intuitions described above. Note that an increased index corresponds to points earlier in time. The definition of the sets S_i, R_i, A_i, A_i^r is quite intuitive: S_i contains the preconditions of the actions in the previous A_i, and R_i contains the preconditions of actions in A_i^r. A_i^r contains actions that have a useful, but not mutually exclusive, effects. A_i is defined much like A_i^r, but w.r.t. S_i rather than R_i. The set CS_i contains literals that are mutually exclusive at a particular point. A set L of literals is mutually exclusive if any set of relevant actions that have L among their preconditions are mutually exclusive. The set CA_i contains mutually exclusive sets of actions. A set of actions A is mutually exclusive if it contains interfering actions or if the set of its effects is mutually exclusive.

Example: In order to illustrate the Relevant-k algorithm, we shall once again use the counter example used in Section 3.1, starting with a three bit counter and using the propositions, p_1, p_2, p_3. Each of the actions a_1, a_2, a_3 can change the value of a single bit from 0 to 1, provided the values of the lower bits are 1.

We start with the final state $\{\neg p_1, \neg p_2, p_3\}$ and $k = 1$. Since the final state is fully specified, there is no distinction between the sets S_i and R_i and between A_i^r and A_i. A_0 contains the action a_3 and the three relevant noops. S_1 contains $\{p_1, \neg p_1, p_2, \neg p_2, p_3, \neg p_3\}$, A_1 now contains a_1, a_2, a_3, and the appropriate noops, and the remaining sets look the same.

If $k = 2$, A_0 and S_1 are as in the $k = 1$ case. However, CS_1 contains $(\neg p_1, p_2)$ and $(p_1, \neg p_2)$, which implies that a_2 cannot be applied. Hence, A_1 contains a_1 and a_3, but not a_2, unlike the case of $k = 1$. The action a_2 would be introduced only in the next step.

Next, consider a partially specified goal, such as $\{p_3, p_2\}$ and with $k = 2$. $A_0^r = \{a_2, \text{NOOP}[\text{P}_2], \text{NOOP}[\text{P}_3]\}$ because a_2 has p_2 as an effect, and a_2 does not destroy p_3; whereas a_1, for example, does not have an effect in the goal. A_0 would now contain A_0^r as well as the noops for p_1 and $\neg p_1$. $R_1 = \{p_1, p_2, \neg p_2, p_3\}$ and $S_1 = \{p_1, \neg p_1, p_2, \neg p_2, p_3\}$. Next, A_1^r contains $\{a_1, a_2\}$, etc.

Finally, suppose we have four bits, and the goal state is $\{\neg p_1, \neg p_2, p_3, p_4\}$ (i.e., the counter's bit value is 1100). If $k = 2$, A_0 contains a_3 and $S_1 = \{p_1, \neg p_1, p_2, \neg p_2, p_3, \neg p_3, p_4\}$. However, CS_1 contains pairs such as $(\neg p_1, \neg p_3), (\neg p_2, \neg p_3)$ and others. A_1 contains a_1, a_3 and some noops. $S_2 = S_1$, but now, CS_2 does not contain $(\neg p_1, \neg p_3)$, it does contain $(\neg p_2, \neg p_3)$, though, and that precludes action a_4 from being in A_2. In the next step, we have $S_3 = S_2 = S_1$, and CS_3 no longer contains $(\neg p_2, \neg p_3)$. This implies that we can add a_4 to A_3 because its effects are no longer mutually exclusive. So overall, we have $A_0 = \{a_3\}, A_1 = \{a_1, a_3\}, A_2 = \{a_1, a_2, a_3\}$, and $A_3 = \{a_1, a_2, a_3, a_4\}$. However, if $k = 3$, at

CS_3 we would still have a mutual exclusion constraint on $(\neg p_1, \neg p_2, \neg p_3)$, which would not allow us to add a_4. Hence, when $k = 3$, $A_3 = \{a_1, a_2, a_3\}$. ∎

We can prove the following soundness results:

Theorem 2 *Let s be some state from which the goal is reachable using an m-step plan (where each step can contain a number of non-interfering actions). Then (1) the set of literals satisfied in s is a subset of S_m, no subset of which is in CS_m, and (2) there exists an m-step plan for reaching the goal from s such that if A is the set of actions in the plan v steps before last then $A \subset A_v$ and no subset of A is in CA_v.*

A corollary of this theorem is:

Corollary 1 *For any initial state s from which the goal is reachable and any minimal (in the number of operators) plan $P = A'_m, \ldots A'_0$ (where the steps are numbered backwards) for reaching the goal from s, we have $A'_i \subset A_i$ and CA_i does not contain any subset of A'_i.*

The complexity of Relevant-k is $O(|A|^k|L| + |L|^k m_p^k |A|)$, where $|A|$ is the number of actions, $|L|$ is the number of proposition in the language, and m_p is the maximal number of preconditions of an action. For more details, see Appendix B.

We now compare the amount of relevance information that can be propagated backwards using k-clause resolution and the goal literals as opposed to Relevant-k. Consider unit propagation first. In the context of the linear encoding, we see that all actions that destroy some goal condition will be ruled out. However, actions that are irrelevant because they produce irrelevant effects will not be pruned.[6] On the other hand, Relevant-1 prunes both actions that destroy some goal literal and actions that are simply irrelevant. There is a slightly degenerate case in which all actions but one destroy some goal proposition. In that case, using unit propagation we will be able to deduce the previous state. Consequently, we have:

Lemma 4 *In the context of the linear encoding, unless there is a single safe, final action, unit propagation yields less relevance information than Relevant-1.*

In the context of the Graphplan encoding the situation is often worse, and unit propagation prunes even less than in the linear encoding. The goal propositions appear only in class 2 (effect) axioms. Propagating them against these axioms, we obtain disjunctions of positive action propositions explaining a particular goal proposition. If we assume that all literals have more than one explanation, we see that no new unit clauses emerge. Consequently, we can prune nothing.

Example: Consider the blocks' world domain once again. Suppose that there are three blocks A,B, and C, and that the goal is ON(A,B). Clearly, any action that moves block C or moves another block on top of block C is irrelevant as a last action. When we consider the Graphplan encoding, the only unit clause we have is ON(A,B,t) (where t is the last time point). We can resolve it against the effect axiom that lists the the possible causes for ON(A,B,t). Aside from the noop action, there are actions such as moving A from C to B

---

[6] In general, proving that an action should be ruled out means that we have shown that in all models, i.e., all plans, this action does not appear. We cannot expect to be able to do this for an irrelevant action since it could possibly be inserted into the plan without affecting it.

and moving A from the Table to B. This yields a new ternary clause and no additional unit clauses. There are no other axioms in which ON(A,B,t) appears negated.

Notice that we have no means of excluding actions that destroy one of the goal literals. For example, if our goal was CLEAR(A,t), we would not want the action MOVE(B,C,A,t-1) as a last action. However, as above, all that we can deduce from CLEAR(A,t) is: MOVE(B,A,C,t-1)$\vee$MOVE(B,A,Table,t-1)$\vee$ MOVE(C,A,B,t-1)$\vee$MOVE(C,A,Table,t-1)$\vee$ NOOP[CLEAR$(A, t{-}1)$]. If we could use binary resolution at this stage, we could deduce the negation of any action with the effect $\neg$CLEAR(A,t), because any such action would be mutually exclusive from any of the above five actions. ▮

If a goal literal l has a single explanation it must be a noop action (which implies that there is no "real" operator that has it as an effect). In that case, we would be able to deduce that this noop action must hold, and using the precondition axioms, we would deduce that l must hold at the previous step. Using the mutex axioms (class 4) we could deduce the negation of any action that destroys l. However, we cannot deduce the negation of any action that does not interact with l, whether it is simply irrelevant or it destroys some other goal literal.

Example: Consider a domain such as the Rocket domain, where a rocket can have fuel, but there is no action for fueling a rocket. Suppose that the rocket has fuel in the goal state. Hence, FUEL(t) holds. Since the explanation axiom for FUEL is a binary clause (i.e., $\neg$FUEL$(t)\vee$NOOP[FUEL(T $-$ 1)]. Resolving this axiom with the fact FUEL(t), we derive a new unit clause NOOP[FUEL$(t-1)$]. Using the precondition axioms, we can derive FUEL(t-1). Using the mutex axiom, we can derive an action such as FLY(t-1), one of whose effects is $\neg$FUEL$(t-1)$. Notice, though, that we cannot deduce the negation of an action that does not interact with the proposition FUEL, whether or not it is irrelevant. For example, if FUEL is the only proposition in the goal, then an action such as loading the rocket, which does not affect the value of the proposition FUEL need not be considered for the final action of the plan. However, as before, there is no way of deducing $\neg$LOAD$(t-1)$.

Because no action can produce FUEL the same reasoning would apply to any step, and we will be able to deduce the fact that FUEL holds at each time point during the plan. Using this fact, we will be able to prune out all actions that have $\neg$FUEL as a precondition. Relevant-1 will not be able to do so: If a has $\neg$FUEL as a precondition but a has an effect that is relevant at some point, a will be considered a relevant action.[7]

Lemma 5 *In the context of the* GRAPHPLAN *encoding, if there is an action for changing the value of every literal, then unit propagation yields less relevance information than Relevant-1.*

Some actual values appears in Section 5. In particular, in the examples we looked at, the GRAPHPLAN encoding could not prune any action. This follows from the (quite typical) fact that in these domains, each of the facts that hold at the final state can be achieved by a number of actions. Hence, unit propagation can deduce only disjunctions of possible

---

[7] Of course, in this particular domain we do not have an action whose precondition is $\neg$FUEL, but the observation is still valid. For example, we may have a maintenance action which can be performed only when the rocket is without fuel.

actions, none of which are a unit clause. Since we have no way of deducing negated actions, propagation stops at this point.

The general case is similar. In the linear encoding, having obtained a disjunction of allowable actions, we can generate a disjunction of allowable preconditions. This information is propagated backwards much like the forward case. Yet, as in the $k = 1$ case, all we can expect is a form of backwards reachability analysis from the goal state, rather than true relevance analysis. Again, Relevant-k is likely to do a much better job here, because it takes explicit relevance issues into account. However, as in the case of reachability analysis, because of the ability of k-clause resolution to yield constraints of order greater than k, we cannot show that Relevant-k is always better.

In the context of the Graphplan encoding, we will generate disjunctions of relevant actions, from which disjunctions of relevant preconditions can be deduced, etc. However, irrelevant actions will not be excluded explicitly (since more than one action is allowed at each step) and we will only conclude that some relevant action must appear. Nor can we exclude actions that destroy a goal proposition. Again, because we can deduce constraints of order greater than k via k-clause resolution, we cannot provide a general result here.

Finally, we note that (1) the Graphplan planner does not incorporate relevance analysis, but Mea-Graphplan, a more recent variant, does (Kambhampati et al., 1997), as well as IPP (Nebel et al., 1997). (2) Ernst, Millstein, and Weld (1997) discuss an enhanced version of the Graphplan encoding which contains effects axioms as well (i.e., axioms of the form action → effect). In terms of the ability to propagate reachability and relevance information, we see an added ability to rule out actions that destroy needed propositions (as in the linear encoding.)

5. Empirical Evaluation

In the previous sections we attempted to understand the mechanisms by which resolution yields reachability and relevance information and to compare them to a natural class of direct reachability and relevance algorithms. As we noted, the relationship is not always that of subsumption, and it is of interest to examine the actual pruning abilities of these algorithms. In this section we describe the performance of these algorithms on a number of standard planning problems. Because of the limited number of domains used, caution should be exercised in interpreting these results. However, some interesting results emerge.

Our first set of experiments examined the performance of unary methods on large blocks world and logistics domain problems. We used the blocks' world problems bw-dir.a/b/c/d from the Satplan distribution[8] involving 9/11/15/19 blocks, respectively, and (minimal) plans of length 6/9/14/18. The logistics' domain problems are based on instances described in (Brafman & Hoos, 1999) involving 8 packages and 3 cities, with minimal plans of size 6/10/16, respectively. SAT-encodings were generated using the Medic program (Ernst et al., 1997). We used the *crse* options to obtain a linear encoding and the *erpe* options to obtain a Graphplan-like encodings. However, the encoding obtained via the *erpe* options contain explicit effect axioms, as in the linear encoding. These axiom improve the Graphplan-encoding's ability to propagate relevance information.

---

[8] These instances are part of the UCPOP distribution, maintained by the University of Washington, or from http://www.research.att.com/ kautz/blackbox/index.html, the BlackBox home page.

	$\|A\|$	Reach	Rel	R+R	U-rch(l)	U-rel(l)
log.a	4565	2922	617	3476	401	38
log.b	5941	3517	680	3905	442	20
log.c	8021	5051	2782	6214	600	32
bw.a	3888	1697	408	2105	639	300
bw.b	10890	3565	830	4395	1201	440
bw.c	44100	12818	2394	15212	3141	840
bw.d	116964	26963	5238	32201	6482	5114

Table 1: **Pruning Effects of Unary Methods.** $|A|$ is the number of possible actions in the course of a minimal length plan. The following entries hold the number of actions pruned using: Reachable-1, Relevant-1, both combined, unit propagation on linear encoding using initial state, and using the final state. Unit propagation in the GRAPHPLAN encoding using the final state yielded no pruning. Execution times for the Reach/Relevant algorithms are ≤ 0.01 seconds except for bw.c (0.03 sec.), and bw.d (0.07 sec.).

In this set of experiments we measured the number of potential actions eliminated by the following algorithms: Reachable-1, Relevant-1, Reachable-1 and Relevant-1 combined, reachability analysis via unit-resolution using the initial state, and relevance analysis via unit-resolution using the goal state. We did not consider the GRAPHPLAN encoding for the following reasons:
(1) Unit-propagation in the GRAPHPLAN encoding yields as much information as Reachable-1. (2) For our particular experiments (and in most other cases), unit-resolution based on the final state in the GRAPHPLAN encoding prunes little, if any, actions because for each fact appearing in the goal state there are a number of potential producing actions. (3) The version of the GRAPHPLAN-encoding produced by MEDIC is basically equivalent to the linear-encoding in terms of relevance information because it contains explicit effect axioms.

The actual numbers appear in Table 1. The first column provides the size of the set of actions for the minimal plan length. The following columns provide the number of actions pruned by the various methods tested. It is evident that Reachable-1 is extremely effective. Relevance analysis seems much less useful, although Relevant-1 does prune a non-negligible number of actions. The results for unit-resolution are quite disappointing, although in line with our theoretical analysis. Recalling that unit-resolution in the GRAPHPLAN encoding is equivalent to Reachable-1, we see that there is a much greater potential for pruning in the GRAPHPLAN encoding. Another interesting observation is that there is little overlap between the reachability and relevance analysis. This stems from the fact that the pruning effect of these algorithms is often quite shallow: most of the pruning is done on the very first steps (in reachability) or very last steps (in relevance). Finally, we note that the $k = 1$ algorithms are quite fast: Unit propagation is an important heuristic in all SAT solution algorithms based on the David-Putnam algorithm (Davis & Putman, 1960), and it is extremely fast, with negligible running times (i.e., < 0.01 seconds). Not surprisingly,

	$\|A\|$	Rch1	Rch2	Rel1/2	rch1(l)	rch2(l)	rch1(gp)	rch2(gp)	rel1/2(l/g)
bw-sm.a	18	21	22	8	15	22	21	22	4
bw-sm.b	48	68	70	44	44	74	68	70	6
bw-sm.c	100	199	204	184	96	210	199	204	12
log-sm.a	18	39	57	8	14	49	39	44	1
log-sm.b	42	111	165	18	36	141	111	126	3
log-sm.c	66	196	292	26	58	244	196	220	5
hanoi-3	38	94	97	21	36	117	94	118	9
hanoi-4	68	224	230	34	66	281	224	280	14
hanoi-5	110	450	460	50	108	558	450	551	20

Table 2: **Effects of Unary and Binary Methods.** $|A|$ is the number of possible actions *per step*. The following columns hold the number of actions pruned during the course of a minimal-length (or longer) plan using Reachable-1, Reachable-2, Relevant-1 and 2 (which yield the same value), unit propagation on the linear encoding using initial state, binary propagation on the linear encoding using initial state, unit propagation on the Graphplan encoding using initial state, and binary propagation on the Graphplan encoding using initial state. The final column correspond to propagation using the goal state. All methods (i.e., unit and binary) on both encodings yielded the same values.

Reachable-1 and Relevant-1 are also extremely fast. Execution times for these algorithms were less than ≤ 0.01 seconds, except for bw.c (0.03 sec.), and bw.d (0.07 sec.), for which these amount to a small fraction of the running times required by modern SAT algorithms.[9]

The next set of experiments, shown in Table 2, introduces binary pruning methods as well. Here, we were limited by the slow performance of our prolog implementation of Reachable-2 and the Medic encoder (Ernst et al., 1997). We looked at blocks world problems involving 3,4, and 5 blocks, respectively, and we looked at logistics domain problems involving one package and two cities, three packages and two cities, and three packages and three cities. In addition, we looked at three hanoi-tower problems with 3,4, and 5 disks.

There are a number of points worth mentioning:

- In two domains (blocks' world and hanoi), Reachable-2 is only slightly more useful than Reachable-1. In the logistics domain, on the other hand, Reachable-2 is much more effective. However, we must remember that Reachable-2 yields mutual exclusion constraints which we did not measure. These constraints can be quite useful and they have an important role in the Graphplan planner.
- No clear winner emerges. In the blocks-world domain, binary resolution in the linear encoding prunes more than Reachable-2, whereas in the logistics domain, Reachable-2 prunes more. Interestingly, binary resolution in the Graphplan-encoding is less

---

[9] These experiments were conducted on a PC with a PentiumII-200 processor.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Reach-1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Reach-2	1	2	2	3	3	3	4	4	4	4	5	5	5	5	5	6
U-Res(lin)	1	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
B-Res(lin)	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	9
U/B-Res(gp)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Table 3: **Reachability Analysis in a 16-bit Counter.** Shown are the number of unpruned actions per time step. Rows correspond to Reachable-1, Reachable-2, unit resolution on the linear encoding, binary resolution on the linear encoding. The last row corresponds to unit and binary resolution on the Graphplan encoding, which had identical effect.

effective than in the linear encoding. However, the Graphplan-encoding allows for shorter plans, and consequently, smaller search spaces. Therefore, the Graphplan-encoding is still likely to be more efficient.

- Relevant-2 has no advantage over relevant-1. In fact, this behavior was observed when using resolution as well: unit and binary resolution on both the linear and Graphplan encodings pruned the same amount of actions. Consequently, we present them in one column. Indeed, we see from both sets of experiments reported in Tables 1 and 2, that relevance analysis contributes little. One obvious reason is that the goal state is often incomplete and much less constrained than the initial state (at least explicitly). Therefore, the algorithms have difficulty deriving relevance constraints. However, one's intuition seems to indicate that this should not be the case, at least not to the extent observed. There should be means of providing better relevance analysis, although they may require more sophisticated derivation of state constraints.
- As predicted, relevance analysis is much more useful at the state-space level than at the truth-assignment level.
- As expected, the Graphplan encoding is typically better than the linear encoding.

Finally, we ran some tests on a 16 bit version of the counter domains described in the text. This is a very constrained domain in which only a single action is applicable at each state and we wanted to see how much of this would be discovered by the algorithms. The results are shown in Tables 3 and 4, where the number of permisable actions is given as a function of the the time step. Table 3 presents the results for forward pruning using Reachable-1, Reachable-2, and unit and binary propagation using the Graphplan and linear encodings. Table 4 presents the results for backward pruning using Relevant-1, Relevant-2, and unit and binary propagation using the linear encodings.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Rel1/2,U/B-Res(gp)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
U-Res(l)	1	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
B-Res(l)	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	9

Table 4: **Relevance Analysis in a 16-bit Counter.** Shown are the number of unpruned actions per time step. The (identical) results for Reachable-1, Reachable-2, unit-resolution on the Graphplan encoding, and binary-resolution on the Graphplan encoding appear in the first row. The next rows correspond to unit and binary resolution on the linear encoding, respectively.

6. Conclusion

We have shown a connection between the scheme used to encode planning instances and the ability to propagate reachability and relevance information from the initial and final steps to other time points. We hope that these results will serve to improve our understanding of the factors contributing to the performance of different encoding methods. In addition, we provided a crisp and general formulation of a class of reachability and relevance algorithms that appear in various forms in different planning algorithms. We compared the pruning ability of resolution-based propagation methods which operate on encoded plans, to that of the Reachable-k and Relevant-k algorithms which operate at the plan level. Our empirical results show a complex picture, where no clear winner emerges. However, it seems that when the domain is constrained (making parallel actions less useful) binary methods have little advantage over unary methods. In addition, they show that relevance analysis is best conducted at the plan level. For SAT-based planning algorithms, this would suggest the use of a simple plan-level relevance analysis stage prior to the plan encoding. This observation is confirmed by recent results reported by Do, Srivastava, and Kambhampati (2000).

In (Brafman, 1999), we pointed out that binary clauses form a large fraction of the clauses in SAT-encoded planning problems. Given our results regarding the utility of binary resolution, a natural idea is to augment standard clause simplification techniques (e.g., unit propagation) with some limited form of binary clause preprocessing. Initial results presented there indicated the utility of this idea: In instances where unit clauses could be derived from this form of binary resolution, nice reductions in running time were demonstrated. When unit clauses were not derivable via this method, only a small overhead was incurred. A more principled, systematic, and efficient technique based on these ideas is investigated in (Brafman, 2000).

This work is among the first attempts to theoretically analyze different encoding schemes. We have concentrated on one particular aspect of such encodings, i.e., their ability to propagate concrete state information backwards and forwards. Naturally, this attempt is a-priori limited in its scope, as this ability is only one factor influencing the performance of various algorithms, and its influence is probably more significant in systematic methods based on the David-Putnam procedure than in methods based on stochastic local search.

Other authors have considered some of the ideas presented here, too. Kautz and Selman (1999) discuss the relation between Graphplan's mutex constraint and a restricted form of binary propagation. In particular they show that mutex computation is a limited form of negative binary propagation. In mutex propagation, two assertions of mutual exclusion yield a new one. Of course, each mutual exclusion statement is equivalent to a binary clause (e.g., either action a is not performed or action b is not performed), hence we can view this process as a limited form of binary propagation: From $\{\neg p \vee \neg q\}$ and $\{p \vee \neg r\}$ deduce $\{\neg q \vee \neg r\}$. Graphplan performs this operation, but in an incomplete manner. In addition, they tested additional limited inference methods such as the failed literal strategy (attempting to prove that a particular literal is inconsistent using unit propagation) and the binary failed literal strategy (attempting to prove that a binary clause is inconsistent using unit propagation). These methods do not directly correspond to the methods considered in this paper. More closely related is one of the options in the Medic system for encoding planning problems: a simple inference method which is referred to as simple data-flow analysis (Ernst et al., 1997). This method is basically an instance of Reachable-1.

Haslum and Geffner (2000) present a parametrized class of admissible heuristics functions H^k. There is an interesting and important relation between the heuristic function generation technique discussed in that paper and the parameterized class of reachability analysis algorithms discussed in this paper. When a heuristic function assigns ∞ to some state s this means it believes that goal is not reachable from s. If the heuristic function is admissible, then in fact, this is true. Thus, admissible heuristic functions provide a sound tool for pruning – the goal is not reachable from any state to which they assign the value ∞. In fact, the derivation of the heuristic functions of class H^k is closely related to our computation of Reachable-k. In both cases, instead of analyzing actual states, we analyze subsets of states of size k and their interactions. However, in designing heuristic functions, a greater emphasis is put on the distance from the current state to a state in which some set of literals appears without mutual exclusion constraints (i.e., the indices of the sets S_i and CS_i).

Finally, a recent paper by Do, Srivastava, and Khambhampati (2000) examines encoded planning problems generated by the Blackbox planner. Blackbox utilizes mutual exclusion constraints derived from Graphplan's planning graph. The authors show that these constraints are useful, despite the fact that they increase the size of the encoding. In addition, the authors examine the utility of adding explicit mutual-exclusion constraints stemming from (state-space based) relevance analysis. These constraints appear to improve the planner's performance. In fact, it seems that the constraints described by Do, Srivastava, and Kambhampati (2000) are more powerful than those generated by Relevant-2. We believe that Relevant-k can and should be strengthened, and we hope to examine this issue more closely in the future.

Acknowledgments

I wish to thank Craig Boutilier and Chris Geib for valuable discussions on reachability analysis and the anonymous reviewers for very useful and detailed comments. I am particularly grateful to Olga Rozenfeld who implemented the algorithms in Prolog, suggested the use of the *counter* example for illustrating the algorithms, and provided important corrections to previous drafts. This work was supported in part by the Paul Ivanier Center for Robotics Research and Production Management.

Appendix A. Proofs

Theorem 1 *If a set of propositions or actions is excluded by Reachable-k at time j then there is no feasible plan in which, at time j, these propositions hold, or, respectively, these actions appear.*

Proof: This is immediate: Consider any valid plan and the states of the world during the execution of this plan. It is straightforward to show that both appear within the sets A_i and S_i without being constrained by virtue of this being a valid plan. ▌

Lemma 1 *In the context of the linear encoding, Reachable*-1 yields more reachability information than unit propagation.*

Proof: Given the definitions used earlier on, a more formal statement of this lemma is as follows: Let k be some integer denoting the length of a plan. Let $A_{reach-1}$ be the set of actions pruned by Reachable*-1 up to the k-th level given some planning domain and initial state. Let A_{u-res} be the set of actions that are pruned by unit-resolution on the linear encoding of this planning domain using k steps (i.e., actions for which we can deduce a unit clause containing the negation of their corresponding variable), but without a goal state supplied. Then $A_{reach-1} \supseteq A_{u-res}$, and for some planning instances $A_{reach-1} \supset A_{u-res}$.

First let us consider unit resolution. The unit clauses that are available initially correspond to the propositions that hold at the initial state. The only axioms in which propositions denoting the state at time 0 appear are those of class 1 (precondition axioms) and 3 (frame axioms). However, the clauses in class 3 are ternary and contain at most one such proposition. These ternary frame clauses can yield a unit clause only if we are able to rule out all actions but one, which we cannot, at this stage. Therefore, unit clauses can only be derived by resolving the current unit clauses with class 1 clauses. Such resolutions can yield new unit clauses containing negated actions. These negated actions can be resolved against clauses containing positive action variables. Such variables appear only in class 4 (at-least-one-action) axioms.

Now there are two cases to consider. First, suppose that we have been able to rule out all actions but one. Using the frame and effect axioms, we can derive the state at time 1. Our situation now is analogous to that in which we were at time 0 with knowledge of the initial state. Since Reachable*-1 puts us in the same position, our claim follows (using a simple inductive argument). Next, suppose that we cannot rule out all actions but one. In that case, we have no new unit clauses, and so unit propagation stops. Reachable*-1 will be able to rule out all actions ruled out by the unit propagation process. Moreover, if all the actions that are not ruled out have some common effect, that effect can be deduced using Reachable-*-1, and it can rule out actions that require its negation as a precondition. This type of information is not obtained via unit propagation. ▌

Lemma 3 *In the context of the* Graphplan *encoding, unit propagation and Reachable-1 rule out the same sets of actions, if we ignore the explicit constraints appearing in axiom class 4. If we use these constraints, unit propagation can yield more reachability information.*

Proof: First, suppose we ignore the mutex axioms of class 4. Using unit propagation, we deduce the negation of those actions whose preconditions are violated at time 0. Negated action literals can be resolved against class 2 (effect explanation) axioms. If we have been able to rule out all explanations of some time 1 proposition, we can deduce its negation in this manner. The same mechanism will allow us to exclude this variable when using Reachable-1. Similarly, negated action literals can be resolved against class 3 (at-least-one-action) axioms, but this yields no more information. Those time 1 variables we can deduce can be used to rule out time 1 actions.

Notice the following. If we can deduce p at time 1, then one of the actions that produce p must hold at time 0. This information is not explicit in the Reachable-1 algorithm (although it appear in the Graphplan's planning graph in the form of edges). However, it cannot be used to rule out other actions if we are restricted to unit resolution.

Class 4 axioms can make a difference in the above case. Suppose we have been able to conclude that a particular action a that produces p must occur (i.e., by deducing p and ruling out all its causes except a). In that case, all actions that are mutually exclusive with a cannot occur. These actions may not affect p at all, and their negation need not necessarily be derivable using Reachable-1. ∎

Theorem 2 *Let s be some state from which the goal is reachable using an m-step plan (where each step can contain a number of non-interfering actions). Then (1) the set of literals satisfied in s is a subset of S_m, no subset of which is in CS_m, and (2) there exists an m-step plan for reaching the goal from s such that if A is the set of actions in the plan v steps before last then $A \subset A_v$ and no subset of A is in CA_v.*

Proof: Recall that we assume that any proposition appearing in the effects of an action appears in its preconditions as well. We can always enforce this requirement by converting an action that does not satisfy it into an a set of actions that satisfy it.

Our proof proceeds by induction on the number of steps by which the goal is reachable. Let S be some state from which the goal G is reachable by a single step. Let A be the set of actions in such a one-step plan for reaching G from S. By definition, A does not contain interfering actions. In addition, we know that if G is reachable from S by performing A then the preconditions of A and $G \setminus \mathit{Effects\text{-}Of}(A)$ must hold in S.

First, suppose to the contrary that for some literal $l \in S$, we have that $l \notin S_1$. Notice that by definition of A_0, we have that S_1 contains all literals that are consistent with G. Therefore, l must be inconsistent with G, i.e., $\neg l \in G$. Since $l \in S$, there must be some action $a \in A$ with the precondition l and the effect $\neg l$ (otherwise, l would hold after performing A). Such an action would be in A_0^r and its preconditions, l among them, would be in S_1. We conclude that $S \subseteq S_1$.

Next, we want to show that there is a one-step plan for reaching G from S all of whose actions are in A_0. From the discussion above we see that the plan A for reaching G from

S contains an action from A_0^r for changing the value of every proposition l that holds in S and that is inconsistent with G. Clearly, none of these actions can have an effect that is inconsistent with G. Let $A' \subseteq A$ denote the set of such actions. By applying A' at S we transform all literals inconsistent with G to their value in G and we do not destroy the value of any literal consistent with G. Since $A' \subseteq A$, it constitutes a valid plan (i.e., its actions do not interfere with each other) that achieves G. By definition, $A' \subseteq A_0$.

To conclude the proof of the base step, we must show that no subset of S is in CS_1. Suppose, to the contrary that some subset S' of S is in CS_1. We have seen that for any such S', there is some set of actions $A' \subset A$ such that $A' \subset A_0$ and each $l \in S'$ is either a precondition of some action in A' or l is consistent with G and is not destroyed by A'. Denote by A'' the set consisting of A' and any NOOP[] corresponding to those $l \in S'$ that are not preconditions of an element in A'. By definition of A_0, we have that $A'' \in A_0$.

However, if $S' \in CS_1$ then $A'' \in CA_0$ which implies that A'' contains interfering actions. We claim that this is impossible. First, all the effects of A'' are either in G or consistent with G, by construction. In addition, all the preconditions of A'' are in S' and therefore in S. Because S is an actual state of the world, it cannot contain conflicting literals. Hence, $S' \notin CS_1$.

Next, suppose that we have established our inductive hypothesis for all $i < m$ and let us prove that it holds for $i = m$. Hence, let S be some state for which there exists an m-step plan $A = A_1, \ldots, A_m$ for attaining G. Let S^{+1} denote the state obtained by applying A_1 to S. We know that there is an $m-1$ step plan for achieving G from S^{+1}. By our inductive hypothesis, S^{+1} satisfies the conditions of the Theorem. In particular, we know that $S^{+1} \subset S_{m-1}$ and no subset of S^{+1} is in CS_{m-1}. To complete our proof it would be sufficient to show that S^{+1} is reachable by a one-step plan A' whose actions are in A_m but not in CA_m. The proof is similar to the base case. ▮

Corollary 1 *For any initial state s from which the goal is reachable and any minimal (in the number of operators) plan* $P = A'_m, \ldots A'_0$ *(where the steps are numbered backwards) for reaching the goal from s, we have* $A'_i \subset A_i$ *and* CA_i *does not contain any subset of* A'_i.

Proof: An inspection of the proof of the previous theorem shows that in every step we have found some subset of the set of actions in each candidate plan that satisfied the relevant conditions. In particular, consider a minimal plan, all its elements must satisfy these conditions. ▮

Lemma 4 *In the context of the linear encoding, unless there is a single safe, final action, unit propagation yields less relevance information than Relevant-1.*

Initially, our only unit clauses are goal literals. We can resolve then against the effect axioms only. This would yield negation of various actions (i.e., unsafe actions). These negated action literals can be resolved only against the action disjunction (axiom class 4). However, if there is more than one safe final action, we will not obtain a unit clause from this disjunction, and there is nothing farther that we can do. The same information, and more, is easily obtainable from Relevant-1. ▮

Lemma 5 *In the context of the* GRAPHPLAN *encoding, if there is an action for changing the value of every literal, then unit propagation yields less relevance information than Relevant-1.*

Proof: See text prior to this Lemma. ▮

Appendix B. The Complexity of Reachable-k and Relevant-k

The computational complexity of Reachable-k is $O(n|A||L|^k E^k + n|L||A|^k)$, where n is the number of levels we generate, $|A|$ is the number of possible actions, $|L|$ is the size of the propositional language used, and E is the maximal number of actions that have a particular shared effect. As we explain below, the complexity is dominated by the time required to produce the sets CS_i and CA_i.

The set of possible effects, S_i, is produced in $O(|A| \cdot m_e)$ steps, where m_e is the maximal number of effects.

CS_i requires examining all l-tuples of elements in S_i, for $l \leq k$, and there are at most $O(|L|^k)$ such elements. For each such tuple we have to find the set of actions that produce it. This can be done quickly, provided we maintained pointers to these actions. The number of such sets of actions is $O(E^K)$ (since no more than k actions are needed). For each such set of actions we must check whether some subset of it is a member of CA_{i-1}. Given an appropriate representation of CA_{i-1}, this can be done in time $O(|A|)$. To accomplish this, we can use a binary tree whose leafs correspond to bit vectors. The depth of this tree is $|A|$ and its size is $O(|CA_{i-1}|)$. Finally, we need to maintain CS_i as a similar tree of bit-vectors. This can be done in $O(|L|^k)$ (or, if CS_i is small, at a lower cost). The overall cost of producing CS_i is $O(|L|^k|E|^k|A|)$.

To produce the set A_i, we go over all actions and check whether their preconditions appear in S_i. This requires $O(|A| \cdot m_p)$ steps (assuming a bit-vector representation of S_i), where m_p is the maximal number of preconditions of an action. We also have to check whether the preconditions appear in CS_i. Since $|A_i| \leq |A|$ and we can check whether a subset of the set of preconditions appears in CS_i in time $O(|L|)$, this requires $O(|A||L|)$ steps.

Finally, we need to produce the CA_i. This requires generating all subsets of A_i of size k or less, taking $O(|A|^k)$ steps. For each such subset we must check whether its preconditions contain an element of CS_i. Again, provided an appropriate data-structure for CS_i is maintained, this can be done in $O(|L|)$ for each set of preconditions. As in the case of CS_k, we assumed CA_i is maintained as a tree of bit-vectors, which can be generated in time $O(|A|^k)$. The overall complexity of this step is $O(|L||A|^k)$.

Note that for small values of k other data-structures are likely to provide better performance.

Next, we address Relevant-k. Our analysis is under the assumption that the same set of variables appear in the preconditions and effects of each operator. As we noted, transforming a set of operators that do not satisfy this property into a set of operators that satisfy it may cause an exponential blow-up in the worst case.

The complexity of Relevant-k is $O(|A|^k|L| + |L|^k m_p^k |A|)$, where $|A|$ is the number of actions, $|L|$ is the number of proposition in the language, and m_p is the maximal number of preconditions of an action. The analysis is quite similar to the case of Reachable-k, and we ignore the sets R_i and A_i^r which are subsets of the larger S_i and A_i and whose generation contributes constant factors:

The set of preconditions, S_i, is produced in $O(|A| \cdot m_p)$ steps.

To compute CS_i, we iterate over $O(|L|^k)$ sets of literals. For each such set we examine all sets of actions that have it as preconditions, and there are at most $O(m_p^k)$ such sets. For each such set of actions, we need to check that it is not in CA_{i-1}. Each such check can be performed in $O(|A|)$ steps. The overall complexity of this step is $O(|L|^k m_p^k |A|)$.

To produce the set A_i, we go over all actions useful for S_i, which require $O(|L|E)$ (where as before, E is the maximal number of actions that have a particular effect). For each action, we check whether its effects are in CS_i. Since we need to perform this check at most once for every action, the overall complexity of $O(|L|E + |L||A|)$.

Finally, we need to produce the sets CA_i. Interfering actions can be pre-computed with the cost amortized over all steps. In any case, their computation requires no more than $O(m|A|^2)$ steps, where m is the maximal sum of preconditions and effects for an action. Next, we have to examine the effects of all l-tuples of actions, where $l \leq k$, and see whether these effects have a subset in CS_i. This takes $O(|A|^k|L|)$ steps.

Again, for small values of k (and in particular, $k = 1, 2$) a tighter analysis is possible.

References

Bayardo, R. J., & Schrag, R. C. (1997). Using CSP look-back techniques to solve real-world SAT instances. In *Proc. AAAI-97*, pp. 203–208.

Blum, A., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*, 281–300.

Bonet, B., Loerincs, G., & Geffner, H. (1997). A robust and fast action selection mechanism for planning. In *Proc. AAAI-97*, pp. 714–719.

Boutilier, C., Brafman, R. I., & Geib, C. (1998). Structured reachability analysis for markov decision processes. In *Proc. of 14th Conference on Uncertainty in AI*, pp. 24–32.

Boutilier, C., & Dearden, R. (1994). Using abstractions for decision theoretic planning with time constraints. In *Proc. of AAAI'94*, pp. 1016–1022.

Brafman, R. I. (1999). Reachability, relevance, resolution, and the planning as satisfiability approach. In *IJCAI'99*, pp. 976–981.

Brafman, R. I., & Hoos, H. H. (1999). To encode or not to encode - i: linear planning. In *IJCAI'99*, pp. 988–993.

Brafman, R. I. (2000). A simplifier for propositional formulas with many binary clauses. Tech. rep. 00-04, Dept. of Computer Science, Ben-Gurion University.

Crawford, J., & Auton, L. D. (1993). Experimental results on the cross-over point in satisfiability problems. In *Proc. AAAI'93*, pp. 21–27.

Davis, M., & Putman, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, *7*, 201–215.

Do, M. B., Srivastava, B., & Kambhampati, S. (2000). Investigating the effect of relevance and reachability constraints on sat encodings of planning. In *Proc. of the Fifth Intl. Conf. on AI Planning and Scheduling Systems*.

Ernst, M. D., Millstein, T. D., & Weld, D. S. (1997). Automatic SAT-compilation of planning problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Fikes, R., & Nilsson, N. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(3–4), 189–208.

Freeman, J. W. (1995). *Improvements to Propositional Satisfiability Search Algorithms*. Ph.D. thesis, U. Pennsylvania Dept. of Computer and Information Science.

Genesereth, M. R., & Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*. Kaufmann, Los Altos, CA.

Gomes, C. P., Selman, B., & Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proc. of 15th Nat. Conf. AI*, pp. 431–437.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In *Proc. of the Fifth Intl. Conf. on AI Planning and Scheduling Systems*, pp. 140–149.

Kambhampati, S., Parker, E., & Lambrecht, E. (1997). Understanding and exending graphplan. In *Proc. 4th European Conf. on Planning*, pp. 260–272.

Kautz, H., & Selman, B. (1992). Planning as satisfiability. In *Proc. of the 10th European Conf. on AI*, pp. 359–363.

Kautz, H., & Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. of the 13th National Conference on AI (AAAI'96)*, pp. 1194–1201.

Kautz, H., & Selman, B. (1999). Unifying sat-based and graph-based planning. In *Proc. 16th Intl. Joint Conf. on AI (IJCAI'99)*, pp. 318–325.

Li, C. M., & Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. In *Proc. IJCAI-97*.

McDermoot, D. (1996). A heuristic estimator for means-ends analysis in planning. In *Proc. 3rd Int. Conf on AI Planning Systems*, pp. 142–149.

Nebel, B., Dimopoulos, Y., & Koehler, J. (1997). Ignoring irrelevant facts and operators in plan generation. In *Proc. 4th European Conf. on Planning*.