

Recent Advances in AI Planning

Daniel S. Weld

■ The past five years have seen dramatic advances in planning algorithms, with an emphasis on propositional methods such as GRAPHPLAN and compilers that convert planning problems into propositional conjunctive normal form formulas for solution using systematic or stochastic SAT methods.

Related work, in the context of spacecraft control, advances our understanding of interleaved planning and execution. In this survey, I explain the latest techniques and suggest areas for future research.

The field of AI planning seeks to build control algorithms that enable an agent to synthesize a course of action that will achieve its goals. Although researchers have studied planning since the early days of AI, recent developments have revolutionized the field. Two approaches, in particular, have attracted much attention: (1) the two-phase GRAPHPLAN (Blum and Furst 1997) planning algorithm and (2) methods for compiling planning problems into propositional formulas for solution using the latest, speedy systematic and stochastic SAT algorithms.

These approaches have much in common, and both are affected by recent progress in constraint satisfaction and search technology. The current level of performance is quite impressive, with several planners quickly solving problems that are orders of magnitude harder than the test pieces of only two years ago. As a single, representative example, the BLACKBOX planner (Kautz and Selman 1998a) requires only 6 minutes to find a 105-action logistics plan in a world with 10^{16} possible states.

Furthermore, work on propositional planning is closely related to the algorithms used in the autonomous controller for the National Aeronautics and Space Administration (NASA) *Deep Space One* spacecraft, launched in October 1998. As a result, our understanding of interleaved planning and execution has advanced

as well as the speed with which we can solve classical planning problems.

The goal of this survey is to explain these recent advances and suggest new directions for research. Because this article requires minimal AI background (for example, simple logic and basic search algorithms), it's suitable for a wide audience, but my treatment is not exhaustive because I don't have the space to discuss every active topic of planning research.¹ I progress as follows: The remainder of the introduction defines the planning problem and surveys freely downloadable planner implementations. The next sections discuss GRAPHPLAN, SAT compilation, and interleaved planning and execution. I conclude by quickly mentioning other recent advances and suggestion topics for future research.

Preliminaries

A simple "classical" formulation of the planning problem defines three input: (1) a description of the initial state of the world in some formal language, (2) a description of the agent's goal (that is, what behavior is desired) in some formal language, and (3) a description of the possible actions that can be performed (again, in some formal language). This last description is often called a *domain theory*.

The planner's output is a sequence of actions that, when executed in any world satisfying the initial state description, will achieve the goal. Note that this formulation of the planning problem is quite abstract—in fact, it really specifies a class of planning problems parameterized by the languages used to represent the world, goals, and actions. For example, one might use propositional logic to describe the effects of actions, which would make it quite awkward to describe actions with universally quantified effects, such as a machine shop spray paint action that coats all objects in the hopper. Thus, one might

GRAPHPLAN and Its Descendants

- GRAPHPLAN—The original, somewhat dated, C implementation (Blum and Furst 1995) is still available at www.cs.cmu.edu/afs/cs.cmu.edu/user/avrim/www/graphplan.html.
- IPP (Koehler et al. 1997a) is a highly optimized C implementation of GRAPHPLAN, extended to handle expressive actions (for example, universal quantification and conditional effects). It is available at www.informatik.uni-freiburg.de/~koehler.
- STAN (Long and Fox 1998) is another highly optimized C implementation that uses an in-place graph representation and performs sophisticated type analysis to compute invariants. It is available at www.dur.ac.uk/~dcs0www/research/stanstuff/stanpage.html.
- SGP (Weld, Anderson, and Smith 1998) is a simple, pedagogical Lisp implementation of GRAPHPLAN, extended to handle universal quantification, conditional effects, and uncertainty. See www.cs.washington.edu/research/projects/ai/www/sgp.html.

Systems Based on Compilation to SAT

- The highest performance SAT compiler is BLACKBOX (Kautz and Selman 1998a). It is available at www.research.att.com/~kautz/blackbox/index.html.
- The MEDIC planner is a flexible test bed, implemented in Lisp, allowing direct comparison of more than a dozen different SAT encodings. See ftp.cs.washington.edu/pub/ai/medic.tar.gz.

Figure 1. Several Implemented Planners Are Available for Download.

describe the effects of actions with first-order predicate calculus, which still assumes that all effects are deterministic. In general, there is a spectrum of more and more expressive languages for representing the world, an agent's goals, and possible actions. In this article, I start by explaining algorithms for planning with the STRIPS representation.² The STRIPS representation describes the initial state of the world with a complete set of ground literals. The STRIPS representation is restricted to goals of attainment, which are defined as a propositional conjunction; all world states satisfying the goal formula are considered equally good. A *domain theory* (that is, a formal description of the actions that are available to the agent) completes a planning problem. In the STRIPS representation, each action is described with a conjunctive precondition and a conjunctive effect that define a transition function from worlds to worlds. The action can be executed in any world w satisfying the precondition formula. The result of executing an action in world w is described by taking w 's state description and adding each literal from the

action's effect conjunction in turn, eliminating contradictory literals along the way.

The classical planning problem makes many simplifying assumptions: atomic time, no exogenous events, deterministic action effects, omniscience on the part of the agent, and so on. I relax some of these assumptions later in the article.

Many readers will find it helpful to experiment with implementations of the ideas discussed in this article. Fortunately, there are a variety of freely distributed alternatives (figure 1), and most accept domains expressed in PDDL syntax,³ the language used for the AIPS (AI planning systems) planning competition,⁴ which I expect will be adopted widely as a standard for teaching purposes and collaborative domain interchange for performance comparison.

GRAPHPLAN and Descendants

Blum and Furst's (1997, 1995) GRAPHPLAN algorithm is one of the most exciting recent developments in AI planning for two reasons:

First, GRAPHPLAN is a simple, elegant algorithm that yields an extremely speedy planner—in many cases, orders of magnitude faster than previous systems such as SNLP (McAllester and Rosenblitt 1991), PRODIGY (Minton et al. 1989), or UCPOP (Penberthy and Weld 1992).

Second, the representations used by GRAPHPLAN form the basis of the most successful encodings of planning problems into propositional SAT; hence, familiarity with GRAPHPLAN aids in understanding SAT-based planning systems (see *Compilation of Planning to SAT*). GRAPHPLAN alternates between two phases: (1) graph expansion and (2) solution extraction. The *graph-expansion phase* extends a planning graph forward in “time” until it has achieved a necessary (but possibly insufficient) condition for plan existence. The *solution-extraction phase* then performs a backward-chaining search on the graph, looking for a plan that solves the problem; if no solution is found, the cycle repeats by further expanding the planning graph.

I start our discussion by considering the initial formulation of GRAPHPLAN, thus restricting our attention to STRIPS planning problems in a deterministic, fully specified world. In other words, both the preconditions and effects of actions are conjunctions of literals (that is, positive literals denote entries in the add lists, and negative literals correspond to elements of the delete list). After covering the basics, I describe optimizations and explain how to handle more expressive action languages.

Expanding the Planning Graph

The planning graph contains two types of node—(1) proposition and (2) action—arranged into levels (figure 2). Even-numbered levels contain proposition nodes (that is, ground literals), and the zero level consists precisely of the propositions that are true in the initial state of the planning problem. Nodes in odd-numbered levels correspond to action instances; there is one such node for each action instance whose preconditions are present (and are mutually consistent) at the previous level. Edges connect proposition nodes to the action instances (at the next level) whose preconditions mention these propositions, and additional edges connect from action nodes to subsequent propositions made true by the action’s effects.

Note that the planning graph represents parallel actions at each action level; thus, a planning graph with k action levels can represent a plan with more than k actions. However, just because two actions are included in the planning graph at some level doesn’t mean

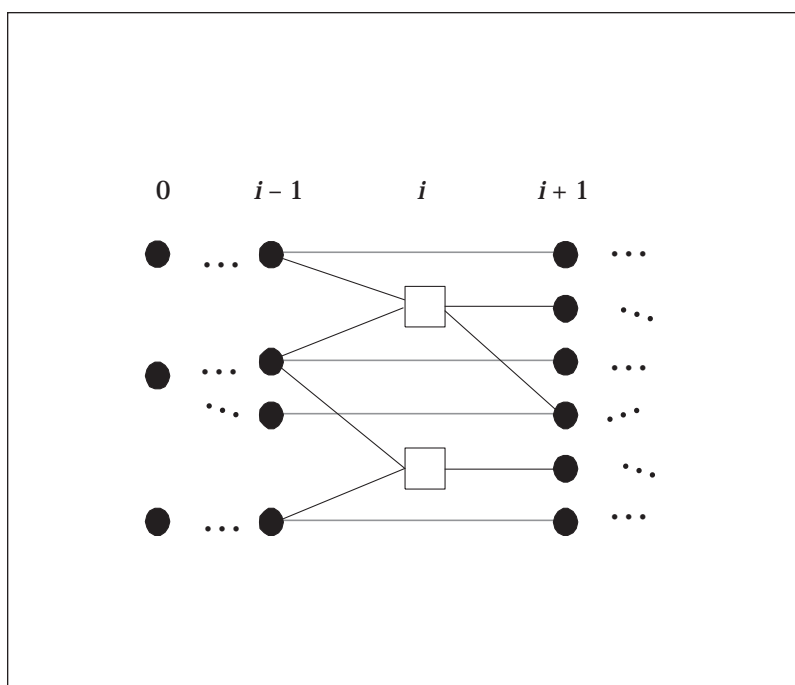


Figure 2. The Planning Graph Alternates Proposition (Circle) and Action (Square) Layers.

Horizontal gray lines between proposition layers represent *maintenance actions*, which encode the possibility that unaffected propositions will persist until the next layer.

that it is always possible to execute both at once. Central to GRAPHPLAN’s efficiency is inference regarding a binary mutual exclusion relation (*mutex*) between nodes at the same level. I define this relation recursively as follows (see also figure 3):

First, two action instances at level i are mutex if either (1) the effect of one action is the negation of another action’s effect (*inconsistent effects*), (2) one action deletes the precondition of another (*interference*), or (3) the actions have preconditions that are mutually exclusive at level $i - 1$ (*competing needs*).

Second, two propositions at level i are mutex if one is the negation of the other or if all ways of achieving the propositions (that is, actions at level $i - 1$) are pairwise mutex (*inconsistent support*).

For example, consider the problem of preparing a surprise date for one’s sleeping sweetheart (figure 4). The goal is to take out the garbage, fix dinner, and wrap a present. There are four possible actions: (1) cook, (2) wrap, (3) carry, and (4) dolly. Cook requires clean hands and achieves dinner. Wrap has the precondition quiet (because the gift is a surprise, one mustn’t wake the recipient) and produces present. Carry eliminates the garbage, but the intimate contact with a smelly container negates clean-

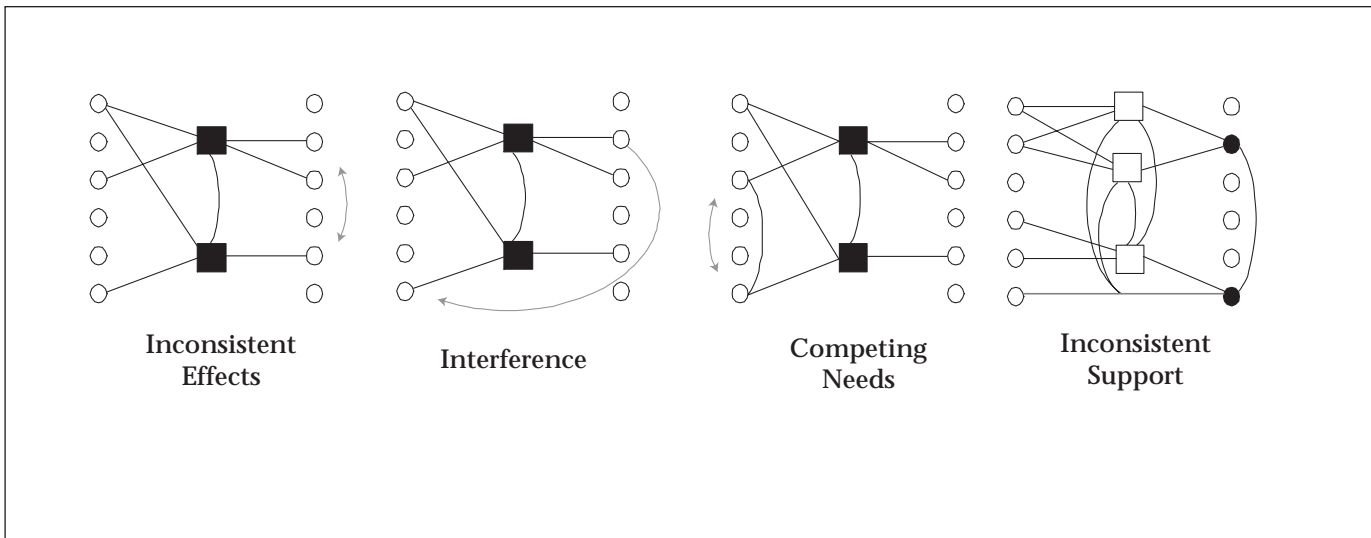


Figure 3. Graphic Depiction of the Mutex Definition (derived by David Smith).

Circles denote propositions; squares represent actions; and thin, curved lines denote mutex relations. The first three parts illustrate deduction of a new action-action mutex (between the dark boxes), and the rightmost part depicts the discovery of a new mutex between propositions (the dark circles).

Hands. The final action, *dolly*, also eliminates the *garbage*, but because of the noisy *handtruck*, it negates *quiet*. Initially, you have *cleanHands*, and the house has *garbage* and is *quiet*; all other propositions are false.

Figure 5 shows the planning graph for the dinner-date problem expanded from level zero through one action and proposition level. Note that the *carry* action is mutex with the persistence of *garbage* because they have inconsistent effects. *Dolly* is mutex with *wrap* because of interference because *dolly* deletes *quiet*. At proposition level two, \neg quiet is mutex with *present* because of inconsistent support. Recall that the goal of the dinner-date problem is to achieve \neg garbage \wedge dinner \wedge present. Because all these literals are present at proposition level two and because none of them are mutex with each other, there is a chance that a plan exists. In this case, the second phase of GRAPHPLAN is executed: solution extraction.

Solution Extraction

Suppose that GRAPHPLAN is trying to generate a plan for a goal with n subgoal conjuncts, and (as in our example) it has extended the planning graph to an even level, i , in which all goal propositions are present, and none are pairwise mutex. This condition is necessary (but insufficient) for plan existence, so GRAPHPLAN performs *solution extraction*, a backward-chaining search to see if a plan exists in the current planning graph.

Solution extraction searches for a plan by considering each of the n subgoals in turn. For

each such literal at level i , GRAPHPLAN chooses an action a at level $i - 1$ that achieves the subgoal. This choice is a backtrack point: If more than one action produces a given subgoal, then GRAPHPLAN must consider all of them to ensure completeness. If a is consistent (that is, nonmutex) with all actions that have been chosen so far at this level, then GRAPHPLAN proceeds to the next subgoal; otherwise, if no such choice is available, GRAPHPLAN backtracks to a previous choice.

After GRAPHPLAN has found a consistent set of actions at level $i - 1$, it recursively tries to find a plan for the set formed by taking the union of all the preconditions of these actions at level $i - 2$. The base case for the recursion is level zero—if the propositions are present there, then GRAPHPLAN has found a solution. Otherwise, if backtracking fails on all combinations of the possible supporting actions for each subgoal (at each level), then GRAPHPLAN extends the planning graph with additional action and proposition levels and then tries solution extraction again.

In the dinner-date example, there are three subgoals at level two: (1) \neg garbage is supported by *carry* and *dolly*, *dinner* is supported by *cook*, and *present* is supported by *wrap*. Thus, GRAPHPLAN must consider two sets of actions at level one—{*carry*; *cook*; *wrap*} and {*dolly*; *cook*; *wrap*}—but unfortunately, neither of these sets is consistent because *carry* is mutex with *cook*, and *dolly* is mutex with *wrap*. Thus, solution extraction fails, and GRAPHPLAN extends the planning graph to level four, as shown in figure 6.

```

Initial Conditions: (and (garbage) (cleanHands) (quiet))
Goal: (and (dinner) (present) (not (garbage)))
Actions:
  cook  :precondition (cleanHands)
        :effect (dinner)
  wrap  :precondition (quiet)
        :effect (present))
  carry :precondition
        :effect (and (not (garbage)) (not (cleanHands)))
  dolly :precondition
        :effect (and (not (garbage)) (not (quiet)))
    
```

Figure 4. STRIPS Specification of the Dinner-Date Problem.

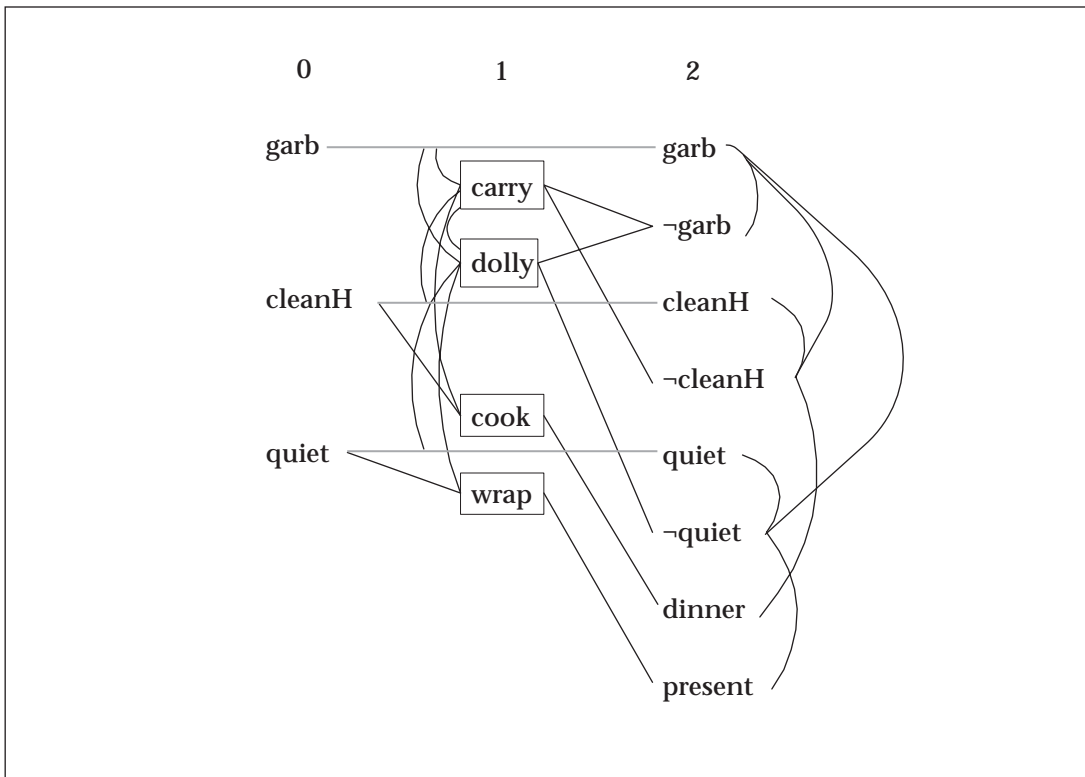


Figure 5. Planning Graph for the Dinner-Date Problem, Expanded Out to Level Two.

Action names are surrounded by boxes, and horizontal gray lines between proposition layers represent maintenance actions that encode persistence. Thin, curved lines between actions and propositions at a single level denote mutex relations.

Note the difference between levels two and four of the planning graph. Although there are no new literals present at level four, there are fewer mutex relations. For example, there is no mutex between dinner and cleanHands at level four. The most important difference is at level three, where there are five additional maintenance actions encoding the possible persistence of literals achieved by level two. Thus, each of the subgoals has additional supporting

actions for consideration during the backward-chaining process of solution extraction. Specifically, \neg garbage is supported by carry, dolly, and a maintenance action. Dinner is supported by cook and a maintenance action. Present is supported by wrap and a maintenance action, so solution extraction needs to consider $3 \times 2 \times 2 = 12$ combinations of supporting actions at level three instead of the $2 \times 1 \times 1 = 2$ combinations during the previous attempt at

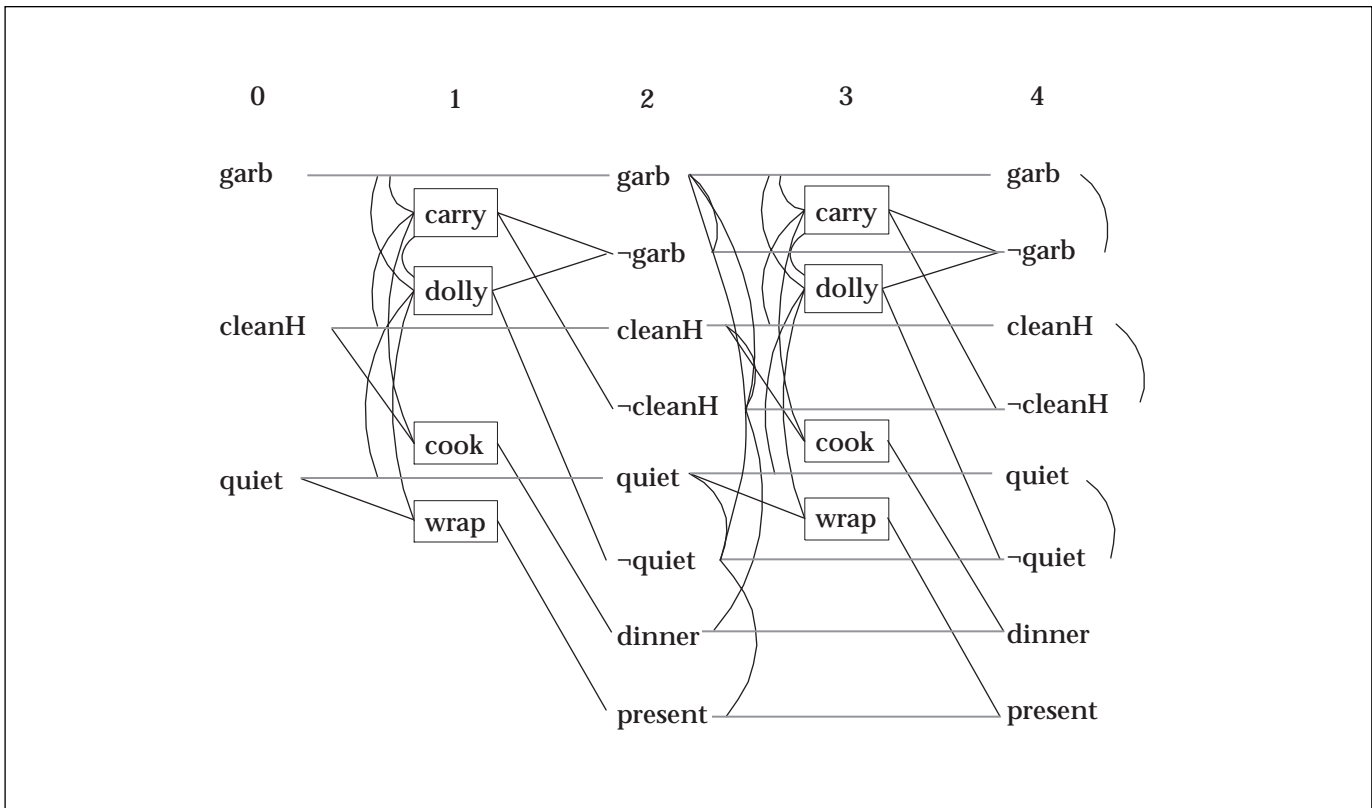


Figure 6. Planning Graph for the Dinner-Date Problem, Expanded to Level Four.

Although no new literals are present at this proposition level, both *dinner* and *present* have additional support from persistence actions, and as a result, GRAPHPLAN's solution-extraction search can find a plan.

solution extraction. Indeed, this increased flexibility allows solution extraction to find a plan. There are actually several combinations that work; I illustrate one below. Support \neg garbage with carry, support dinner with the maintenance action, and support present with wrap. None of these actions is mutex with another, so the choices for level three are consistent. The selection of these actions leads to the following subgoals for level two: dinner (precondition of the maintenance action) and quiet (precondition of wrap); because carry has no preconditions, there are only two level-two subgoals. Solution extraction recurses and chooses cook to support dinner and the maintenance action to support quiet; these two actions aren't mutex, so the selections for level one are consistent. The preconditions of these actions create two subgoals for level zero: cleanHands and quiet. Because these propositions are present in the initial conditions, the selection is consistent, and a solution plan exists!

Figure 7 illustrates the results of solution extraction. Note that GRAPHPLAN generates an inherently parallel (partially ordered) plan. The actions selected for level three, carry and

wrap, can be executed in either order and will achieve the same effect. Thus, if one wishes a totally ordered sequence of actions for one's plan, one can choose arbitrarily: cook, carry, wrap.

Optimizations

To this point, we have covered the basic GRAPHPLAN algorithm, but several optimizations have a huge effect on efficiency. The first improvements speed solution extraction: forward checking, memoization, and explanation-based learning. The second set of optimizations concerns the graph-expansion process: handling of the closed-world assumption, compilation of action schemata to remove static fluents using type analysis, regression focusing, and in-place graph expansion. The benefit achieved by each of these optimizations depends on the specific planning problem to be solved. In the worst case, planning graph expansion is polynomial time, but solution extraction is exponential (Blum and Furst 1995). However, in many planning problems, it is expansion time that dominates, so each of the optimizations described in the following subsections is important.

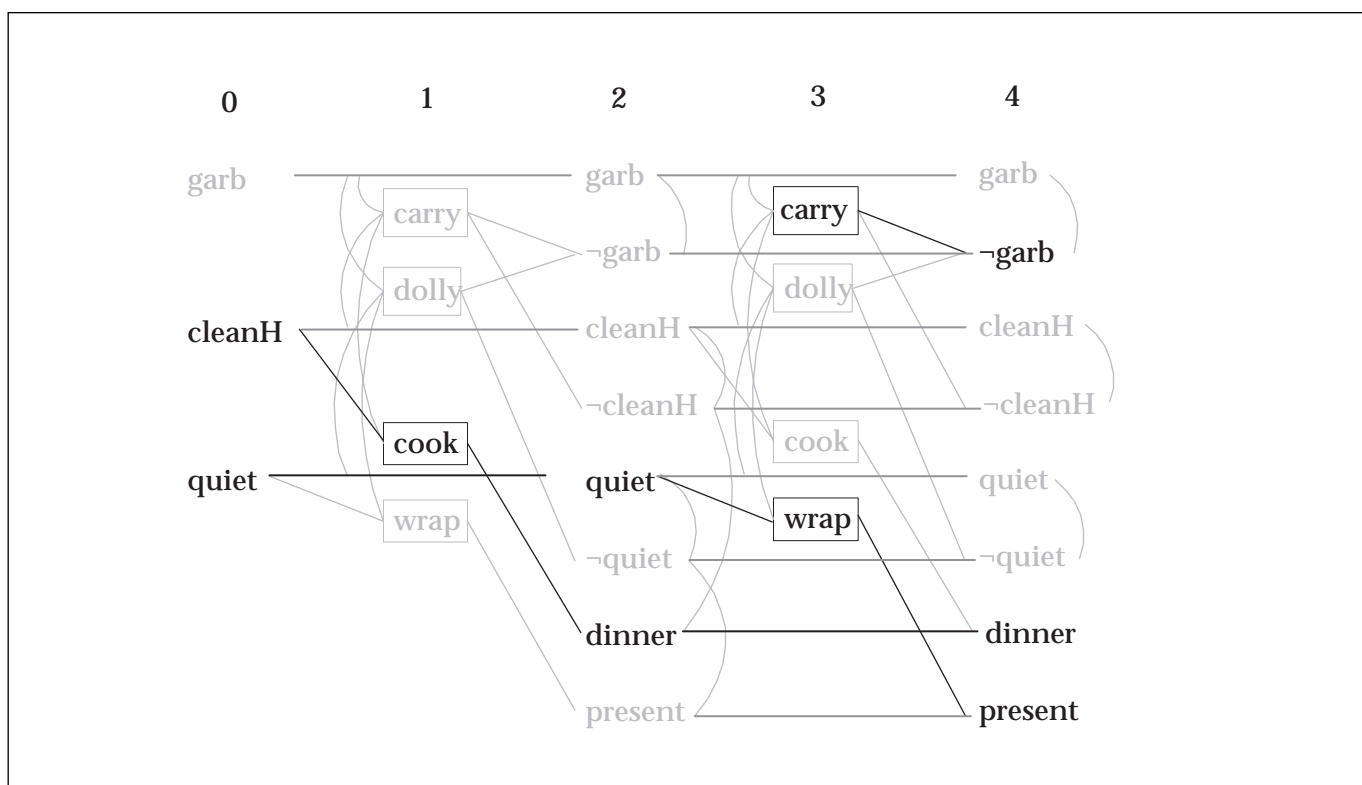


Figure 7. One of Four Plans That Might Be Found by Solution Extraction.

Actions in black are to be executed; all others are not.

Solution Extraction as Constraint Satisfaction

By observing the connection between the GRAPHPLAN solution-extraction process and constraint-satisfaction problems (CSPs), we can transfer many insights from the CSP field to planning.⁵ There are many possible formulations, but the simplest is in terms of a *dynamic CSP* (Falkenhainer and Forbus 1988), that is, a CSP in which the set of variables and associated constraints changes based on the selection of values to earlier variables. There is a CSP variable for subgoal literals at each proposition level after level zero. The *domain* of a variable (that is, its set of possible values) is the set of supporting actions at the previous level. The set of constraints is defined by the mutex relations. For example, consider the process of solution extraction from the level-four dinner-date graph shown in figure 6. Initially, we create a CSP variable for each subgoal at level four: $V_{4, \neg\text{garbage}}$ takes a value from {carry, dolly, maintain}, $V_{4, \text{dinner}}$ takes a value from {cook, maintain}, and $V_{4, \text{present}}$ takes a value from {wrap, maintain}. The assignments $V_{4, \neg\text{garbage}} = \text{carry}$, $V_{4, \text{dinner}} = \text{maintain}$, and $V_{4, \text{present}} = \text{wrap}$ correspond to the first part of the solution shown in figure 7. Once a solution is found for the variables at proposition level four, the actions corresponding to the variable

values define a CSP problem at level two. Note that there is no requirement to perform this search by level. In other words, our previous description of solution extraction dictated finding a consistent set of actions at level i before performing any search at level $i - 2$. However, this methodical order is unnecessary and potentially inefficient. For example, the BLACKBOX planner (Kautz and Selman 1998a) takes the planning graph, compiles it to SAT, and uses fast stochastic methods to perform the equivalent of solution extraction in which search jumps around from level to level in a greedy fashion. Rintanen (1998) describes an opportunistic, nondirectional search strategy that bypasses conversion to SAT.

By itself, this CSP formulation of solution extraction is unremarkable, but it suggests certain strategies for speeding the search, such as forward checking, dynamic variable ordering, memoization, and conflict-directed back-jumping.

When assigning a value to a variable, simple CSP solvers check to ensure that this choice is consistent with all values previously chosen. A better strategy, called *forward checking* (Haralick and Elliott 1980), checks unassigned variables in addition, shrinking their domain by eliminating any values that are inconsistent with

The use of action schemata requires a few changes to the planning graph expansion routine at action levels: The planner must instantiate each parameterized schema to create possible ground actions by considering all combinations of appropriately typed constants.

the recent choice. If the domain of any unassigned variable collapses (that is, it shrinks to the empty set), then the CSP solver should backtrack. Kondrack and van Beek (1997) show analytically that forward checking is an excellent strategy, strengthening previous empirical support.

Dynamic variable ordering refers to a class of heuristics for choosing which CSP variable should next be assigned a value (Bacchus and van Run 1995). Of course, eventually all variables must have values assigned, but the order in which they are selected can have a huge impact on efficiency (Barrett and Weld 1994; Korf 1987). Note that if a variable has only one choice, then it is clearly best to make that assignment immediately. In general, a good heuristic is to select the variable with the fewest remaining (nonconflicting) values, and this information is readily available if forward checking is used.⁶ Although not astounding, these techniques lead to significant (for example, 50 percent) performance improvements in GRAPHPLAN (Kambhampati 1998a). Another method for determining a good subgoal ordering is through structural analysis of subgoal interactions (Cheng and Irani 1989; Smith 1989; Irani and Cheng 1987). Precomputation aimed at calculating speedy subgoal orderings is closely related to the use of abstraction in planning (Knoblock 1991; Yang and Tenenber 1990; Tenenber 1988). In general, one can distinguish between *domain-specific approaches* (which are based on action definitions alone) and *problem-specific approaches* (which additionally use the goal and initial state specifications); problem-specific approaches typically provide more leverage, but the cost of domain-specific precomputation can be amortized among many planning problems. Koehler (1998b) describes a problem-specific method that speeds GRAPHPLAN by orders of magnitude on problems from many domains.⁷

The original GRAPHPLAN paper (Blum and Furst 1995) describes a technique called *memoization*, which caches for future use the results learned from exhaustive search about inconsistent subgoal sets. Suppose that solution extraction is called at level i and, in the course of search, attempts to achieve subgoals P , Q , R , and S at level k (where $k \leq i$). If none of the combinations of supporting actions for these subgoals proves consistent (regardless of the level at which this inconsistency is detected), then GRAPHPLAN records the set $\{P; Q; R; S\}$ to be a nogood at level k . Later, if GRAPHPLAN extends the planning graph to level $i + 2$ and once again attempts solution extraction, it might attempt to achieve the same four goals at level

k , but this time, it will backtrack immediately rather than perform exhaustive search. The memoization process trades space for time, and although the space requirements can be large, the resulting speedups are significant.

As stated, this memoization process is rather simplistic, and because more sophisticated approaches have proven effective in systematic SAT solvers (Bayardo and Schrag 1997), one might suspect memoization improvements are possible. Indeed, recent work by Kambhampati (1998a) demonstrates dramatic speedups (for example, between 1.6 and 120 times faster depending on the domain). The basic idea is to determine which subset of goals is responsible for failure at a given level and record only this subset; if solution extraction ever returns to the level with this set or a superset, then failure is justified. This approach leads to much smaller (and, hence, more general) nogoods; for example, it might be the case that subgoals P , Q , and S are together unachievable, regardless of R .

An additional idea (also described in Kambhampati [1998a]) is the regression of level k failure explanations through the action definitions for level $k + 1$ to calculate failure conditions for level $k + 2$. When these level- $k + 2$ conditions are short, then many searches can be terminated quickly. These methods are based on Kambhampati's (199b) earlier work on the relationship between traditional planning-based speedup methods (for example, explanation-based learning) and CSP methods.

Closed-World Assumption The *closed-world assumption* says that any proposition not explicitly known to be true in the initial state can be presumed false. A simple way of implementing the closed-world assumption in GRAPHPLAN would be to explicitly close the zero level of the planning graph—that is, to add negative literals for all possible propositions that were not explicitly stated to be true. Because there are an infinite number of possible propositions, one should restrict this approach to the relevant subset—that is, those that were mentioned in the preconditions of some action or in the goal; other literals can't affect any solution.

A better solution is to handle the closed-world assumption lazily because this approach shrinks the size of the planning graph and diminishes the cost of graph expansion. Create the zero level of the planning graph by adding only the propositions known to be true in the initial state (as shown in figure 5). When expanding the planning graph to action level i , one does the following: Suppose action A requires $\neg P$ as precondition. If $\neg P$ is in the planning graph at level $i - 1$, simply link to it as

usual. However, if $\neg P$ is missing from level $i-1$, one must check to see if its negation (that is, proposition P) is present at level zero. If P is absent from level zero, then add $\neg P$ to level zero and add maintenance actions and mutexes to carry $\neg P$ to the current level. With this simple approach, no changes are necessary for solution extraction.

Note that the issue of the closed-world assumption never arose with respect to the dinner-date example because none of the actions had a negative precondition. Although the goal did include a negative literal (\neg garbage), the positive proposition *garbage* was present in the initial conditions, thus voiding the closed-world assumption.

Action Schemata, Type Analysis, and Simplification

In the dinner-date example, all the actions were propositional, but in realistic domains, it is much more convenient to define parameterized action schemata. For example, in a logistics domain, one might define the operation of driving a truck, as shown in figure 8. The intuition is simply that at this level of abstraction, driving one vehicle has the same preconditions and effects as driving another; so, one should only write it once.

The use of action schemata requires a few changes to the planning graph expansion routine at action levels: The planner must instantiate each parameterized schema to create possible ground actions by considering all combinations of appropriately typed constants. For example, to handle the drive schema, the system must create $O(n^3)$ ground drive actions, assuming that there are n constants defined in the initial state of the world.

Many of these ground combinations will be irrelevant because the selection of constants to parameters will never satisfy the preconditions. For example, if $?v$ is bound to Seattle, then presumably the ground precondition (vehicle Seattle) will never be satisfied; so, an important optimization involves *type analysis*, which determines which predicates represent types, calculates the set of constants that form the extent of each type, and then instantiates ground actions only for plausibly typed combinations of constants.

The simplest form of type analysis scans the set of predicates present in the initial conditions that are absent from the effects of any action schemata. These predicates (for example, location and vehicle) are static, so terms formed with these predicates (for example, (location Seattle) and (vehicle truck37)) will never change. Thus, the planner can conclude that Seattle is in the extent of the location type and similarly reason about vehicle. Because

```
(defschema (drive)
  :parameters  (?v ?s ?d)
  :precondition (and (vehicle ?v)
                    (location ?s)
                    (location ?d)
                    (road-connected ?s ?d)
                    (at ?v ?s))
  :effect      (and (not (at ?v ?s))
                    (at ?v ?d)))
```

Figure 8. Parameterized Specification of the Action of Driving a Vehicle from a Source Location to a Destination.

one can evaluate static terms at instantiation time, there is no need to include these terms in the planning graph at all; action schemata that include these terms as preconditions can be simplified (during instantiation) by eliminating static preconditions. Furthermore, this simplification is not limited to unary predicates. For example, if vehicle, location, and road-connected are all static and if instances of these actions are only instantiated for constants that obey these preconditions, then planning graph expansion can add ground action instances (such as the one shown in figure 9) and eliminate static terms from proposition levels in the planning graph.

Fox et al. have devised more sophisticated, polynomial-time, planner-independent type-inference methods that deduce state invariants, and they demonstrate that this analysis can significantly speed their version of GRAPHPLAN on some domains (Fox and Long 1998).⁸ Their method is based on the observation that a planning domain can be viewed as a collection of finite-state machines where domain constants traverse between states corresponding to predicates.

Regression Focusing As described previously, the planning graph with d proposition levels contains only those actions that could possibly be executed in the initial state or in a world reachable from the initial state.⁹ However, many of the actions in the planning graph can be irrelevant to the goal at hand. In other words, the graph-expansion algorithm is *uninformed* by the goal of the planning problem, and as a result, time can be wasted by adding useless actions and their effects into the graph and reasoning about mutex relations involving these irrelevant facts.

Two optimizations have been proposed to make graph expansion more goal directed: (1)

```

drive-truck37-Seattle-Tacoma
:precondition (at truck37 Seattle)
:effect      (and (not (at truck37 Seattle))
              (at truck37 Tacoma))

```

Figure 9. Ground Instance of Drive after Type Analysis and Elimination of “Timeless” (Eternally True, Static) Preconditions.

Compare with the schema in figure 8.

heuristically filtering of the facts from the initial state with a fact-generation graph (Nebel, Dimopoulos, and Koehler 1997) and (2) backward expansion of the planning graph (Kambhampati, Lambrecht, and Parker 1997).

A fact-generation graph is an And-Or graph created from a problem goal and domain actions as follows: The root of the graph is an And node corresponding to the goal, and its children are the conjunctive subgoals. Each subgoal P is an Or node whose children correspond to the different ground actions that have P as an effect. This structure would be a tree, except to avoid exponential blowup, nodes are reused within levels of the graph. We say that an Or node is solved if it is in the initial conditions or if an immediate child is solved, and an And node is solved if all its children are solved. Because the fact-generation graph ignores subgoal interactions from negative literals, solution of a depth d fact-generation graph is a necessary but insufficient condition for solution of a depth d planning graph. At the risk of incompleteness, one can try to speed planning by eliminating initial conditions (or ground actions) that don't appear (or appear infrequently) in the fact-generation graph (Nebel, Dimopoulos, and Koehler 1997). Note that this approach is similar to (and motivated by) McDermott's (1996) greedy regression graph heuristic.

A similar approach (Kambhampati, Lambrecht, and Parker, 1997) provides speedup without sacrificing completeness. Recall that GRAPHPLAN follows a simple loop: Expand the planning graph with action and proposition levels, then attempt solution extraction; if no plan is found, then repeat. Kambhampati modified the loop to first grow the planning graph backwards from the subgoals by an action and proposition level, then grow the graph forwards from the intersection of the initial state and the backward propositional fringe, including only ground actions that

were added during backwards propagation and adding in mutex relations, then performing solution extraction if necessary. If solution extraction failed to find a plan, then Kambhampati's system would grow the graph backwards for another pair of levels, compute a new (larger) intersection with the initial state and resume forward growth. Although Kambhampati's implementation regenerated the graphs from scratch at each stage (duplicating discovery of mutex relations), the resulting planning graph was so much smaller that his system outperformed the basic GRAPHPLAN on most problems.

In-Place Graph Expansion One can avoid duplicated work during regression focusing by exploiting the following observations concerning monotonicity in the planning graph:

Propositions are monotonically increasing: If proposition P is present at level i , it will appear at level $i + 2$ and in all subsequent proposition levels.

Actions are monotonically increasing: If action A is present at level i , it will appear at level $i + 2$ and in all subsequent action levels.

Mutexes are monotonically decreasing: If mutex M between actions A and B is present at level i , then M is present at all previous action levels in which both A and B appear. The same is true of mutexes between propositions.¹⁰

Nogoods are monotonically decreasing: If subgoals P , Q , and R are unachievable at level i , then they are unachievable at all previous proposition levels.¹¹

These observations suggest that one can dispense with a multilevel planning graph altogether. Instead, all one needs is a bipartite graph with action and proposition nodes. Arcs from propositions to actions denote the precondition relation, and arcs from actions to propositions encode effects. Action, proposition, mutex, and nogood structures are all annotated with an integer label field; for proposition and action nodes, this integer denotes the first planning graph level at which the proposition (or action) appears. For mutex or nogood nodes, the label marks the last level at which the relation holds. By adding an additional set of labels, one can interleave forward and backward expansion of the planning graph. With this scheme, the time and space costs of the expansion phase are vastly decreased, but the bookkeeping required is surprisingly tricky; see Smith and Weld (1998b) for details, and see also the STAN planner's “wave-front” representation (Long and Fox 1998).

```

(defschema (drive)
  :parameters (?v ?s ?d)
  :precondition (and (vehicle ?v) (at ?v ?s)
                    (location ?s) (location ?d)
                    (road-connected ?s ?d))
  :effect (and (at ?v ?d) (not (at ?v ?s))
              (when (in cargo ?v)
                    (and (at cargo ?v) (not (at cargo ?s))))
              (when (in spare-tire ?v)
                    (and (at spare-tire ?d) (not (at spare-tire ?s)))))

```

Figure 10. Conditional Effects Allow the Same Drive Schema to Be Used When the Vehicle Is Empty or Contains Cargo or a Spare Tire.

Handling Expressive Action Languages

Until now, our discussion has been restricted to the problem of planning with the STRIPS representation in which actions are limited to quantifier-free, conjunctive preconditions and effects. Because this representation is severely limited, this subsection discusses extensions to more expressive representations aimed at complex, real-world domains. I focus on disjunctive preconditions, conditional effects, and universally quantified preconditions and effects because these areas have received the most attention. Koehler (1998a) has developed methods for handling resource constraints, and I discuss work on uncertainty at the end of this article (after describing methods for compiling planning problems to SAT). However, other capabilities such as domain axioms, procedural attachment, numeric fluents, exogenous events, and actions with temporal duration beg for exploration.

Disjunctive Preconditions It is easy to extend GRAPHPLAN to handle disjunctive preconditions. Conceptually, the precondition (which might contain nested Ands and Ors) is converted to disjunctive normal form (DNF). Then when the planning graph is extended with an action schema whose precondition contains multiple disjuncts, an action instance can be added if any disjunct has all its conjuncts present (nonmutex) in the previous level. During the solution-extraction phase, if the planner at level i considers an action with disjunctive preconditions, then it must consider all possible precondition disjuncts at level $i - 1$ to ensure completeness.

Disjunctive effects are much harder because they imply nondeterminism—one cannot predict the precise effect of execution in advance. As a result, they require a general approach to uncertainty, which I discuss near the end of this article.

Conditional Effects Conditional effects are used to describe actions whose effects are context dependent. The basic idea is simple: We allow a special when clause in the syntax of action effects. When takes two arguments: (1) an antecedent and (2) a consequent; execution of the action will have the consequent's effect just in the case that the antecedent is true immediately before execution (that is, much like the action's precondition determines if execution itself is legal—for this reason, the antecedent is sometimes referred to as a secondary precondition [Pednault 1989]). Note also that like an action precondition, the antecedent part refers to the world before the action is executed, and the consequent refers to the world after execution. For now, we assume that the consequent is a conjunction of positive or negative literals. Figure 10 illustrates how conditional effects allow one to define a single action schema that accounts for driving a vehicle that can possibly contain a spare tire or cargo.

Three methods have been devised for allowing GRAPHPLAN-derivative planners to handle action schemata with conditional effects: (1) full expansion (Gazen and Knoblock 1997), (2) factored expansion (Anderson, Smith, and Weld 1998), and (3) partially factored expansion (Koehler et al. 1997a). The simplest approach, *full expansion*, rewrites an action schema containing conditional effects into a number of mutually exclusive STRIPS schemata by considering all minimal consistent combinations of antecedents in the conditional effects. For example, the action schema in figure 10 would be broken into four separate STRIPS schemata (as shown in figure 11): one for the empty vehicle, one for the vehicle with cargo, one for the vehicle with spare tire, and one for the vehicle with both cargo and spare.

Although full expansion has the advantage of simplicity, it can result in an exponential explosion in the number of actions. If a spare

```

(defschema (drive-empty)
  :parameters (?v ?s ?d)
  :precondition (and (vehicle ?v) (at ?v ?s)
                    (location ?s) (location ?d)
                    (road-connected ?s ?d)
                    (not (in cargo ?v)) (not (in spare-tire ?v)))
  :effect (and (at ?v ?d) (not (at ?v ?s))))
(defschema (drive-cargo)
  :parameters (?v ?s ?d)
  :precondition (and (vehicle ?v) (at ?v ?s)
                    (location ?s) (location ?d)
                    (road-connected ?s ?d)
                    (in cargo ?v) (not (in spare-tire ?v)))
  :effect (and (at ?v ?d) (not (at ?v ?s))
              (and (at cargo ?v)) (not (at cargo ?s))))
(defschema (drive-spare)
  :parameters (?v ?s ?d)
  :precondition (and (vehicle ?v) (at ?v ?s)
                    (location ?s) (location ?d)
                    (road-connected ?s ?d)
                    (not (in cargo ?v)) (in spare-tire ?v))
  :effect (and (at ?v ?d) (not (at ?v ?s))
              (and (at spare-tire ?v)) (not (at spare-tire ?s))))
(defschema (drive-both)
  :parameters (?v ?s ?d)
  :precondition (and (vehicle ?v) (at ?v ?s)
                    (location ?s) (location ?d)
                    (road-connected ?s ?d)
                    (not (in cargo ?v)) (in spare-tire ?v))
  :effect (and (at ?v ?d) (not (at ?v ?s))
              (and (at cargo ?v)) (not (at cargo ?s))
              (and (at spare-tire ?v)) (not (at spare-tire ?s))))

```

Figure 11. The Four STRIPS Schemas for Driving with Possible Contents.

fuel drum could also be in the vehicle, then full expansion would generate eight STRIPS schemata. In general, if an action has n conditional effects, each containing m antecedent conjuncts, then full expansion can produce as many as n^m STRIPS actions (Gazen and Knoblock 1997). This explosion is common when the conditional effects are universally quantified, as in figure 12. In essence, this schema has one conditional effect for each object that could possibly be put in the truck. If there were only 20 such cargo items, full expansion would yield over a million STRIPS schemata.

The other two approaches for dealing with conditional effects consider the conditional effects themselves as the primitive elements handled by GRAPHPLAN.¹² Note that in contrast to the STRIPS actions produced by full expansion, an action's conditional effects are not mutually exclusive, but neither are they independent because the antecedent of one effect can imply that of another. The advantage of factored expansion is an increase in perfor-

mance. By avoiding the need to expand actions containing conditional effects into an exponential number of plain STRIPS actions, factored expansion yields dramatic speedup. However, this increased performance comes at the expense of complexity:

Because factored expansion reasons about individual effects of actions (instead of complete actions), more complex rules are required to define the necessary mutual exclusion constraints during planning graph construction. The most tricky extension stems from the case when one conditional effect is induced by another, that is, when it is impossible to execute one effect without causing the other to happen as well (Anderson, Smith, and Weld 1998).

Factored expansion also complicates the solution extraction because of the need to perform the analog of confrontation (Weld 1994; Penberthy and Weld 1992), that is, prevent the antecedent of undesirable effects from occurring.

The IPP planner (Koehler et al. 1997b) uses a third method for handling conditional effects,

```

(defschema (drive)
  :parameters (?v ?s ?d)
  :precondition (and (vehicle ?v) (at ?v ?s)
                    (location ?s) (location ?d)
                    (road-connected ?s ?d))
  :effect (and (at ?v ?d) (not (at ?v ?s))
              (forall (object ?o)
                (when (in ?o ?v)
                  (and (at ?o ?v) (not (at ?o ?s)))))))

```

Figure 12. Universally Quantified Conditional Schemata for Driving.

which I call *partially factored expansion*. The primary difference stems from IPP's mutex rules that state that two actions are marked as mutex only if their unconditional effects and preconditions are in conflict. This difference allows IPP to do less computation during graph expansion but reduces the number of mutex constraints that will be found. For most domains, the difference doesn't matter, but in some cases (for example, the movie-watching domain [Anderson, Smith, and Weld 1998]), factored expansion performs exponentially better than IPP.

Universal Quantification The GRAPHPLAN descendants IPP (Koehler et al. 1997a) and SCP (Anderson, Smith, and Weld 1998) each allow action schemata with universal quantification. In preconditions, universal quantification lets one conveniently describe real-world actions such as the UNIX `rmdir` command, which deletes a directory only if all files inside it have already been deleted. Universally quantified effects allow one to describe actions such as `chmod *`, which set the protection of all files in a given directory. Naturally, universal quantification is equally useful in describing physical domains. As shown in figure 12, one can use a universally quantified conditional effect to say that all objects on the vehicle will change location as a result of driving.

To add universal quantification to GRAPHPLAN, it helps to make several simplifying assumptions. Specifically, assume that the world being modeled has a finite, static universe of typed objects. For each object in the universe, the initial state description must include a unary atomic sentence declaring its type.¹³ For example, the initial description might include sentences of the form `(vehicle truck37)` and `(location Renton)`, where `vehicle` and `location` are types.¹⁴ The assumption of a

static universe means that action effects might not assert type information. For example, if an action were allowed to assert `(not (vehicle truck37))` as an effect, then it would amount to the destruction of an object; the assumption forbids the destruction or the creation of objects.

To assure systematic establishment of goals and preconditions that have universally quantified clauses, one must modify the graph expansion phase to map these formulas into a corresponding ground version. The *Herbrand base* Υ of a first-order, function-free sentence, Δ , is defined recursively as follows:

$$\Upsilon(\Delta) = \Delta \text{ if } \Delta \text{ contains no quantifiers}$$

$$\Upsilon(\forall_{t1} x \Delta(x)) = \Upsilon(\Delta_1) \dots \Upsilon(\Delta_n)$$

where the Δ_i correspond to each possible interpretation of $\Delta(x)$ under the universe of discourse, $\{C_1; \dots; C_n\}$, that is, the possible objects of type $t1$ (Genesereth and Nilsson 1987). In each Δ_i , all references to x have been replaced with the constant C_i . For example, suppose that the universe of `vehicle` is `{truck37; loader55; plane7}`. If Δ is `(forall ((vehicle ?v)) (at ?v Seattle))`, then the Herbrand base $\Upsilon(\Delta)$ is the following conjunction:

$$\text{(and (at truck37 Seattle) (at loader55 Seattle) (at plane7 Seattle))}$$

Under the static universe assumption, if this goal is satisfied, then the universally quantified goal is satisfied as well. Note that the Herbrand base for a formula containing only universal quantifiers will always be ground, so one can use formulas of this form as action effects. It's easy to handle existential quantifiers interleaved arbitrarily with universal quantification when the expression is used as a goal, action precondition, or the antecedent of a conditional effect. Existential quantifiers are not allowed in action effects because they are equivalent to disjunctive effects and (as

Despite the early formulation of planning as theorem proving ..., most researchers have long assumed that special-purpose planning algorithms are necessary for practical performance.

described earlier) imply nondeterminism and, hence, require reasoning about uncertainty.

To handle existential quantification in goals, one needs to extend the definition of Herbrand base as follows:

$$\begin{aligned} \Upsilon(\exists_{t_1} y \Delta(y)) &= t_1(y) \wedge \Upsilon(\Delta(y)) \\ \Upsilon(\forall_{t_1} x \forall_{t_2} y \Delta(x, y)) &= t_2(y_1) \wedge \Upsilon(\Delta_1) \wedge \dots \wedge t_2(y_n) \wedge \Upsilon(\Delta_n) \end{aligned}$$

Once again, the Δ_i correspond to each possible interpretation of $\Delta(x, y)$ under the universe of discourse for type t_1 : $\{C_1, \dots, C_n\}$. In each Δ_i , all references to x have been replaced with the constant C_i . In addition, references to y have been replaced with Skolem constants (that is, the y_j).¹⁵ All existential quantifiers are eliminated as well, but the remaining free variables (which act as Skolem constants) are implicitly existentially quantified; they will be treated just like action schemata parameters during graph expansion. Because we are careful to generate one such Skolem constant for each possible assignment of values to the universally quantified variables in the enclosing scope, there is no need to generate and reason about Skolem functions. In other words, instead of using $y = f(x)$, we enumerate the set $\{f(C_1), f(C_2), \dots, f(C_n)\}$ for each member of the universe of x and then generate the appropriate set of clauses Δ_i by substitution and renaming. Because each type's universe is assumed finite, the Herbrand base is guaranteed finite as well. Two more examples illustrate the handling of existential quantification: Suppose that the universe of location is $\{\text{Seattle, Renton}\}$ and that Δ is

(exists ((location ?l)
(forall ((vehicle ?v) (at ?v ?l))))

then the Herbrand base is the following:

(and (location ?l) (at truck37 ?l)
(at loader55 ?l) (at plane7 ?l))

As a final example, suppose Δ is

(forall ((location ?l)
(exists ((vehicle ?v) (at ?v ?l))))

Then the universal base contains two Skolem constants ($?v1$ and $?v2$) that are treated as parameters:

(and (vehicle ?v1) (at ?v1 Seattle)
(vehicle ?v2) (at ?v2 Renton))

Because there are only two locations, the Skolem constants $?v1$ and $?v2$ exhaust the range of the Skolem function whose domain is the universe of vehicles. Because of the finite, static universe assumption, one can always do this expansion when creating the Herbrand base.

In summary, we only allow universal quan-

tifiers in action effects, but goals, preconditions, and effect antecedents can have interleaved universal and existential quantifiers. Quantified formulas are compiled into the corresponding Herbrand base, and all remaining variables are treated like action schemata parameters during graph expansion. Because the resulting planning graph contains quantifier-free ground action instances, no changes are required during solution extraction.

Compilation of Planning to SAT

Despite the early formulation of planning as theorem proving (Green 1969), most researchers have long assumed that special-purpose planning algorithms are necessary for practical performance. Algorithms such as TWEAK (Chapman 1987), SNLP (McAllester and Rosenblitt 1991), UCPOP (Penberthy and Weld 1992), and GRAPHPLAN (Blum and Furst 1995) can all be viewed as special-purpose theorem provers aimed at planning problems. However, recent improvements in the performance of propositional satisfiability methods (Cook and Mitchell 1997) call this whole endeavor in doubt. Initial results for compiling bounded-length planning problems to SAT were unremarkable (Kautz and Selman 1992), but recent experiments (Kautz and Selman 1996) suggest that compilation to SAT might yield the world's fastest STRIPS-style planner.

Figure 13 shows the architecture of a typical SAT-based planning system, for example, MEDIC (Ernst, Millstein, and Weld 1997) or BLACKBOX (Kautz and Selman 1998a). The compiler takes a planning problem as input, guesses a plan length, and generates a propositional logic formula, which, if satisfied, implies the existence of a solution plan; a *symbol table* records the correspondence between propositional variables and the planning instance. The simplifier uses fast (linear time) techniques such as unit-clause propagation and pure literal elimination (for example, Van Gelder and Tsuji [1996]) to shrink the CNF formula. The solver uses systematic or stochastic methods to find a satisfying assignment that the decoder translates (using the symbol table) into a solution plan. If the solver finds that the formula is unsatisfiable, then the compiler generates a new encoding reflecting a longer plan length.

The Space of Encodings

Compilers for high-level programming languages (for example, Lisp) are compared on the basis of speed and also on the quality of the machine code they produce. These same notions carry over to SAT compilers as well.

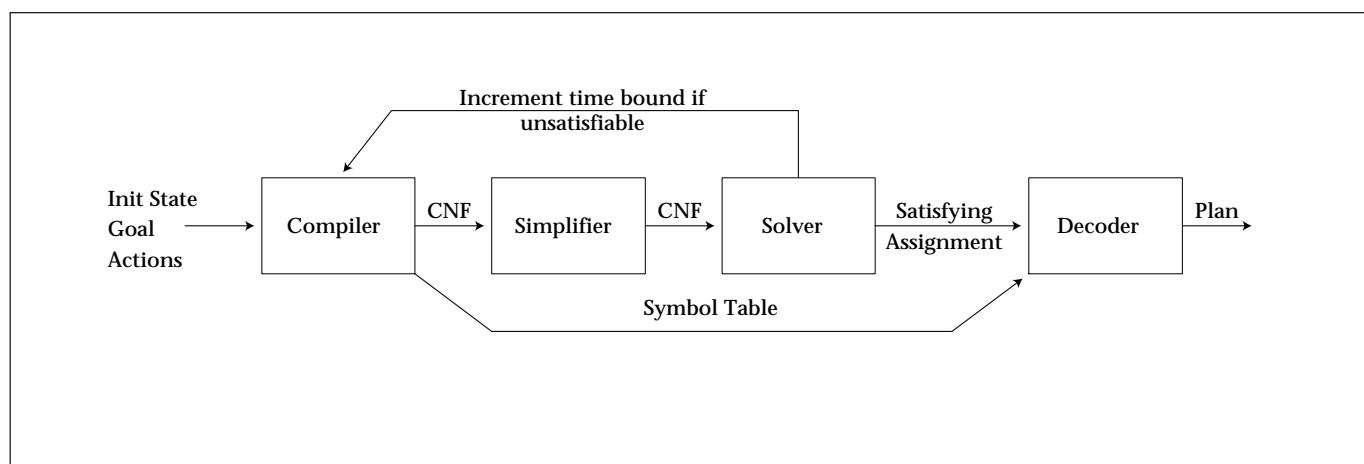


Figure 13. Architecture of a Typical SAT-Based Planning System.

One wants a compiler to quickly produce a small SAT encoding because solver speed can be exponential in the size of the formula being tested. However, this measure of size is complicated by the fact that a propositional formula can be measured in terms of the number of variables, the number of clauses, or the total number of literals summed over all clauses; often a decrease in one parameter (variables, say) will increase another (for example, clauses). Two factors determine these sizes: (1) the encoding and (2) the optimizations being used. Because the encoding is the more fundamental notion, I focus on it first, presenting a parameterized space of possibilities (developed in Ernst, Millstein, and Weld [1997]) with two dimensions:

First, the choice of a regular, simply split, overloaded split, or bitwise *action representation* specifies the correspondence between propositional variables and ground (fully instantiated) plan actions. These choices represent different points in the trade-off between the number of variables and the number of clauses in the formula.

Second, the choice of classical or explanatory *frame axioms* varies the way that stationary fluents are constrained.

Each of the encodings uses a standard fluent model in which time takes nonnegative integer values. State fluents occur at even-numbered times and actions at odd times. For example, in the context of the dinner-date problem described previously, the propositional variable garb_0 means that there is garbage in the initial state, $\neg\text{garb}_2$ signifies that there is no garbage after executing the first set of parallel actions, and carry_1 means that the carry action is executed at time one.

Each of the encodings uses the following set of universal axioms:

INIT: The initial state is completely specified at time zero, including all properties presumed false by the closed-world assumption. For the dinner-date problem, one gets

$$\text{garb}_0 \wedge \text{cleanH}_0 \wedge \text{quiet}_0 \wedge \neg\text{dinner}_0 \\ \wedge \neg\text{present}_0$$

GOAL: To test for a plan of length n , all desired goal properties are asserted to be true at time $2n$ (but the goal state need not be specified fully). Assuming a desired dinner-date plan length of $n = 1$, one gets

$$\neg\text{garb}_2 \wedge \text{dinner}_2 \wedge \text{present}_2$$

A \Rightarrow P,E: Actions imply their preconditions and effects. For each odd time t between 1 and $2n - 1$ and for each consistent ground action, an axiom asserts that execution of the action at time t implies that its effects hold at $t + 1$, and its preconditions hold at $t - 1$. The $A \Rightarrow P,E$ can generate numerous clauses when applied to action schemata in the context of a world with many objects, but for the simple nonparameterized dinner-date *cook* action, one gets

$$(\neg\text{cook}_1 \vee \text{dinner}_2) \wedge (\neg\text{cook}_1 \vee \text{cleanH}_0)$$

Action Representation The first major encoding choice is whether to represent the names of ground action instances in regular, simply split, overloaded split, or the bitwise format. This choice is irrelevant for purely propositional planning problems (such as the dinner-date example) but becomes crucial given parameterized action schemata (for example, the STRIPS drive schema shown in figure 8). In the *regular representation*, each ground action is represented by a different logical variable, for a total of $n \mid \text{Schemata} \mid \text{Dom} \mid P_s$ such variables, where n denotes the number of odd time steps, $\mid \text{Schemata} \mid$ is the number of action schema, P_s denotes the maximum number of parameters for each action schemata, and \mid

Dom | is the number of objects in the domain. Because systematic solvers take worst-case time that is exponential in the number of variables, and large numbers of variables also slow stochastic solvers, one would like to reduce this number.

To reduce the number of variables, Kautz and Selman (1996) introduced simple action splitting, which replaces each n -ary action fluent with n unary fluents throughout the encoding. For example, variables of the form $\text{Drive}(\text{Truck37}, \text{Seattle}, \text{Renton}, t)$ are replaced with the conjunction of $\text{DriveArg1}(\text{Truck37}, t)$, $\text{DriveArg2}(\text{Seattle}, t)$, $\text{DriveArg3}(\text{Renton}, t)$.¹⁶ Replacing variables with the conjunction for each action reduces the number of variables needed to represent all actions to $n | \text{Schemata} || \text{Dom} | P_s$, but each action (formerly a single variable) is now described by a conjunction of P_s variables. With the simple splitting representation, only instances of the same action schemata share propositional variables. An alternative is overloaded splitting, whereby all operators share the same split fluents. Overloaded splitting replaces $\text{Drive}(\text{Truck37}, \text{Seattle}, \text{Renton}, t)$ by the conjunction of $\text{Act}(\text{Drive}, t) \wedge \text{Arg1}(\text{Truck37}, t) \wedge \text{Arg2}(\text{Seattle}, t) \wedge \text{Arg3}(\text{Renton}, t)$, and a different action $\text{Load}(\text{Truck37}, \text{Drum9}, t)$ is replaced with $\text{Act}(\text{Load}, t) \wedge \text{Arg1}(\text{Truck37}, t) \wedge \text{Arg2}(\text{Drum9}, t)$. This technique further reduces the number of variables needed to represent all actions to $n (| \text{Schemata} || \text{Dom} | P_s)$.

The *bitwise representation* shrinks the number of variables even more by representing the action instances with only $\lceil \log_2 | \text{Schemata} || \text{Dom} | P_s \rceil$ propositional symbols (for each odd time step), each such variable representing a bit. The ground action instances are numbered from 0 to $(| \text{Schemata} || \text{Dom} | P_s) - 1$. The number encoded by the bit symbols determines the ground action that executes at each odd time step. For example, if there were four ground actions, then $(\neg \text{bit1}(t) \wedge \neg \text{bit2}(t))$ would replace the first action, $(\neg \text{bit1}(t) \wedge \text{bit2}(t))$ would replace the second, and so on.

Which action representation is the best? Although more experiments need to be performed, preliminary results suggest that the regular and simply split representations are good choices (Ernst, Millstein, and Weld 1997). In contrast, bitwise and overloaded representations result in convoluted encodings that resist simplification and type analysis. For example, although the bitwise encoding yields the smallest number of propositional variables before simplification, the linear-time procedure described in Van Gelder and Tsuji (1996) shrunk the CNF formulas from the other rep-

resentations so much that afterwards, bitwise representations had the most variables.

Frame Axioms Every encoding requires axioms to confront the frame problem (McCarthy and Hayes 1969).

Frame: *Frame axioms* constrain unaffected fluents when an action occurs. There are two alternatives: classical or explanatory frames.

Classical frame axioms (McCarthy and Hayes 1969) state which fluents are left unchanged by a given action. For example, one classical frame axiom for the STRIPS drive schemata (figure 8) would say, "Driving vehicle Truck37 from Seattle to Renton leaves Truck9's location (Kent) unchanged."

$$\begin{aligned} & (\text{At}(\text{Truck9}, \text{Kent}, t - 1) \\ & \wedge \text{Drive}(\text{Truck37}, \text{Seattle}, \text{Renton}, t) \\ & \Rightarrow \text{At}(\text{Truck9}, \text{Kent}, t + 1) \end{aligned}$$

Because the encoding is propositional, one must write a version of this axiom for each combination of (1) possible location of Truck9, (2) source location for Truck37, and (3) destination for Truck37. If these aren't the only two trucks, then there will be even more combinations. Note also the use of the regular action representation implied by our choice of variable $\text{Drive}(\text{Truck37}, \text{Seattle}, \text{Renton}, t)$; if a different representation is desired, then the frame axiom might contain more literals.

Adding classical frame axioms for each action and each odd time t to the universal axioms almost produces a valid encoding of the planning problem. However, if no action occurs at time t , the axioms of the encoding can infer nothing about the truth value of fluents at time $t + 1$, which can therefore take on arbitrary values. The solution is to add at-least-one axioms for each time step.

At-least-one: A disjunction of every possible, fully instantiated action ensures that some action occurs at each odd time step. (A maintenance action is inserted as a preprocessing step.) Note that action representation has a huge effect on the size of this axiom.

The resulting plan consists of a totally ordered sequence of actions; indeed, it corresponds roughly to a linear encoding in Kautz, McAllester, and Selman (1996), except that they include exclusion axioms (see later) to ensure that at most, one action is active at a time. However, exclusion axioms are unnecessary because the classical frame axioms combined with the $A \Rightarrow P, E$ axioms ensure that any two actions occurring at time t lead to an identical world state at time $t + 1$. Therefore, if more than one action does occur in a time step, then either one can be selected to form a valid plan.

Explanatory frame axioms (Haas 1987) enumerate the set of actions that could have

occurred to account for a state change. For example, an explanatory frame axiom would say which actions could have caused truck9 to have left Seattle.

$$\begin{aligned} & (\text{At}(\text{Truck9}, \text{Seattle}, t - 1) \wedge \neg \text{At}(\text{Truck9}, \\ & \text{Seattle}, t + 1)) \Rightarrow \\ & (\text{Drive}(\text{Truck9}, \text{Seattle}, \text{Renton}, t) \vee \\ & \text{Drive}(\text{Truck9}, \text{Seattle}, \text{Kent}, t) \vee \dots \vee \\ & \text{Drive}(\text{Truck9}, \text{Seattle}, \text{Tacoma}, t)) \end{aligned}$$

Note (again) that the choice of action representation affects the length of the frame axioms. Furthermore, note that the axiom can be simplified dramatically if a different representation is chosen. For example, if we use the simply split representation, then a straight translation yields.

$$\begin{aligned} & \text{At}(\text{Truck9}, \text{Seattle}, t - 1) \wedge \neg \text{At}(\text{Truck9}, \\ & \text{Seattle}, t + 1) \Rightarrow ((\text{DriveArg1}(\text{Truck9}, t) \Rightarrow \\ & \text{DriveArg2}(\text{Seattle}, t) \wedge \\ & \text{DriveArg3}(\text{Renton}, t)) \vee \\ & (\text{DriveArg1}(\text{Truck9}, t) \wedge \\ & \text{DriveArg2}(\text{Seattle}, t) \wedge \\ & \text{DriveArg3}(\text{Kent}, t)) \vee \dots \vee \\ & (\text{DriveArg1}(\text{Truck9}, t) \wedge \\ & \text{DriveArg2}(\text{Seattle}, t) \wedge \\ & \text{DriveArg3}(\text{Tacoma}, t))) \end{aligned}$$

However, this disjunction is really just enumerating all the possible destinations, which is silly, so the compiler can do a *factoring optimization* (Ernst, Millstein, and Weld 1997) by recognizing which parameters affect which literals and generating simplified frames axioms.¹⁷ For this example, the compiler should generate (the vastly simpler)

$$\begin{aligned} & \text{At}(\text{Truck9}, \text{Seattle}, t - 1) \wedge \\ & \neg \text{At}(\text{Truck9}, \text{Seattle}, t + 1) \\ & (\text{DriveArg1}(\text{Truck9}, t) \wedge \\ & \text{DriveArg2}(\text{Seattle}, t)) \end{aligned}$$

As a supplement to the universal axioms, explanatory frame axioms must be added for each ground fluent and each odd time t to produce a reasonable encoding. With explanatory frames, a change in a fluent's truth value implies that some action occurs, so (contrapositively) if no action occurs at a time step, the lack of an action occurring will be correctly treated as a maintenance action. Therefore, no at-least-one axioms are required.

The use of explanatory frame axioms brings an important benefit: Because they do not explicitly force the fluents unaffected by an executing action to remain unchanged, explanatory frames permit parallelism. Specifically, any actions whose preconditions are satisfied at time t and whose effects do not contradict each other can be executed in parallel. Parallelism is important because it allows one to encode an n step plan with less than n

odd time steps, and small encodings are good. However, uncontrolled parallelism is problematic because it can create valid plans that have no linear solution. For example, suppose action α has precondition X and effect Y , but action β has precondition $\neg Y$ and effect $\neg X$. Although these actions might be executed in parallel (because their effects are not contradictory), there is no legal total ordering of the two actions. Hence, one must explicitly rule out this type of pathologic behavior with more axioms.

Exclusion: Linearizability of resulting plans is guaranteed by restricting which actions can occur simultaneously.

Two kinds of exclusion enforce different constraints in the resulting plan:

First, with *complete exclusion*, for each odd time step, and for all distinct, fully instantiated action pairs α, β , add clauses of the form $\neg \alpha_t \vee \neg \beta_t$. Complete exclusion ensures that only one action occurs at each time step, guaranteeing a totally ordered plan.

Second, with *conflict exclusion*, for each odd time step, and for all distinct, fully instantiated, conflicting action pairs α, β , add clauses of the form $\neg \alpha_t \vee \neg \beta_t$. In our framework, two actions conflict if one's precondition is inconsistent with the other's effect.¹⁸ Conflict exclusion results in plans whose actions form a partial order. Any total order consistent with the partial order is a valid plan.

Note that conflict exclusion cannot be used in isolation given a split action representation because splitting causes there not to be a unique variable for each fully instantiated action. For example, with simple splitting, it would be impossible to have two instantiations of the same action schema execute at the same time because their split fluents would interfere. Overloaded splitting further disallows two instantiations of different actions to execute at the same time, so it requires complete exclusion. Simple splitting can be used with conflict exclusion when augmented with additional axioms that ban multiple instances of a single schema from executing.

The bitwise action representation requires no action exclusion axioms. At any time step, only one fully instantiated action's index can be represented by the bit symbols, so a total ordering is guaranteed.

What is the best way to represent frame axioms? Experience (Ernst, Millstein, and Weld 1997; Kautz and Selman 1996) shows that explanatory frame axioms are clearly superior to classical frames in almost every case. Because parallel actions encode longer plans with the same number of time steps, conflict exclusion

MEDIC incorporates many switch-selectable optimizations such as type analysis; these features make MEDIC a powerful test bed for research in SAT-based planning.

should be used whenever possible (for example, with the regular action representation or with the minimal additional exclusions necessary for the simply split representation).

Other Kinds of Encoding The MEDIC planning compiler (Ernst, Millstein, and Weld 1997) uses the taxonomy described earlier to generate any of 12 different encodings. In addition, MEDIC incorporates many switch-selectable optimizations such as type analysis; these features make MEDIC a powerful test bed for research in SAT-based planning. However, several encodings do not fit in our taxonomy and, hence, cannot be generated by MEDIC.

The causal encoding (Kautz, McAllester, and Selman 1996) is based on the causal-link representation used by partial-order planners such as SNLP (McAllester and Rosenblitt 1991). Although this encoding has been shown to have the smallest encoding when measured asymptotically, the constant factors are large, and despite several efforts, no one has succeeded in building a practical compiler based on the idea.

Work has also been done on exploring ways of encoding hierarchical task network (HTN) planning (Erol, Hendler, and Nau 1994) as a SAT problem (Mali and Kambhampati 1998).

Comparison with GRAPHPLAN Note the strong similarities between GRAPHPLAN-derivative and SAT-based planning systems: First, both approaches convert parameterized action schemata into a finite propositional structure (for example, the planning graph and a CNF formula) representing the space of possible plans to a given length. Second, both approaches use local consistency methods (for example, mutex propagation and propositional simplification) before resorting to exhaustive search. Third, both approaches iteratively expand their propositional structure until they find a solution.

Indeed, Kautz and Selman (1996) showed that the planning graph can be converted automatically into CNF notation for solution with SAT solvers by constructing propositional formulas stating the following: (1) The (fully specified) initial state holds at level zero, and the goal holds at the highest level (that is, our init and goal axioms). (2) Conflicting actions are mutually exclusive (that is, conflict-based exclusion axioms). (3) Actions imply their preconditions (that is, the precondition part of our $A \Rightarrow P$, E axioms). (4) Each fact at a positive even level implies the disjunction of all actions at the previous level that have this fact in their effects (including a maintenance action if it exists). For example, consider the dinner-date proposition $\neg garb$ at level 4 in figure 6; one obtains

$$\neg garb_4 \Rightarrow (dolly_3 \vee carry_3 \vee maintain\text{-}no\text{-}garb_3)$$

Kautz, McAllester, and Selman (1996) observe that this encoding is close to the combination of explanatory frames with a regular action representation; there are two differences: First, this encoding does not explicitly include explanatory frame axioms, but they can be generated by resolving axioms of type 4 with the “actions imply their preconditions” axioms for the maintenance actions. Second, there are no axioms stating that actions imply their effects, so spurious actions can be included in the solution (these can be removed later by the decoder). Fortunately, the conflict-exclusion axioms prevent these spurious actions from interfering with the rest of the plan.

The BLACKBOX system (Kautz and Selman 1998a) uses this GRAPHPLAN-based encoding to provide a very fast planner. BLACKBOX uses the graph-expansion phase of IPP (Koehler et al. 1997a) to create the planning graph, then converts the graph into CNF rather than perform traditional solution extraction. One of the keys to BLACKBOX’s performance is the observation that the simplification algorithm employed by GRAPHPLAN is more powerful than the unit propagation used in the previous SAT planning system (Kautz and Selman 1998a; Kambhampati 1997a). Specifically, GRAPHPLAN uses negative binary propagation in a limited way: Binary exclusion clauses corresponding to mutex relations (for example, $\{\neg p \vee \neg q\}$) are resolved against proposition support sets (for example, $\{p \vee r \vee s \vee \dots\}$) to infer $\{\neg q \vee r \vee s \vee \dots\}$.

Optimizations

There are several ways to improve the encodings discussed previously; in this subsection, I discuss compile-time-type optimization and the addition of domain-specific information.

The principles and objectives underlying type analysis for SAT compilation are the same as previously discussed in the context of GRAPHPLAN. GRAPHPLAN-based approaches (for example, inertia optimization [Koehler et al. 1997a] and TIM [Fox and Long 1998]) aimed to shrink the size of the planning graph by eliminating static fluents and avoiding nonsensical action-schemata instantiations. The same approaches can be used to shrink the size of the CNF formula that a SAT compiler generates. The MEDIC compiler performs optimizations that reduce CNF size by as much as 69 percent on a variety of problems (Ernst, Millstein, and Weld 1997).

Another way to optimize the CNF formula produced by a compiler is to add domain-specific information. Typically, this knowledge is

impossible to express in terms of STRIPS actions but is natural when writing general logical axioms and can be induced when processing action schemata and initial state specifications. For example, in the blocks world, one might state axioms to the effect that the relation *On* is both noncommutative and irreflexive, only one block can be on another at any time, and so on. Ernst et al. show that adding these types of axiom increased the clause size of the resulting CNF formulas but decreased the number of variables (after simplification) by 15 percent and speed up solver time significantly. Domain axioms can be classified in terms of the logical relationship between the knowledge encoded and the original problem statement (Kautz and Selman 1998b):

First, action conflicts and derived effects are entailed solely by the preconditions and effects of the domain's action schemata.

Second, heuristics that are entailed by the initial state in conjunction with the domain's action schemata include *state invariants*. For example, a vehicle can only be in one location at a time.

Third, optimality heuristics restrict plans by disallowing unnecessary subplans. For example, in a package delivery domain, one might specify that packages should never be returned to their original location.

Fourth, simplifying assumptions are not logically entailed by the domain definition or goal but can restrict search without eliminating plans. For example, one might specify that once trucks are loaded, they should immediately move.

DISCOPLAN (Gerevini and Schubert 1998) is a preprocessing system that infers state constraints from domain definitions. The basic idea is to look for four general axiom patterns (which can be discovered with low-order polynomial effort). For example, a single-valued constraint would be discovered for the logistics world, saying that each vehicle can be in only one place at a time. IPP's planning graph is used to discover some of these constraints, and others are deduced using special-purpose analysis. No attempt is made to deduce optimality heuristics or simplifying assumptions—all constraints are completeness preserving. Nevertheless, the CNF formulas with DISCOPLAN-inferred axioms were solved many times faster than plain MEDIC or SATPLAN (Kautz and Selman 1996) formulas, regardless of the SAT solver used. Indeed, in many cases, the plain encodings were unsolvable in the allotted time, and the DISCOPLAN-augmented encodings quickly yielded a plan.

Other researchers have devised alternative

methods for detecting constraints. For example, Bacchus and Teh (1998) describe a method similar to DISCOPLAN that, in addition, uses regression search to further restrict the predicate domains. Rintanen (1998) modified algorithms from computer-aided verification to discover binary invariants. Earlier work on the subject is presented in Kelleher and Cohen (1992). Despite these promising first efforts, much more exciting work remains to be done in the area of optimizing SAT encodings for speedy solution.

SAT Solvers

Without an efficient solver, a planning-to-SAT compiler is useless; in this subsection, I review the state of the art. Perhaps the best summary is that this area of research is highly dynamic. Each year seems to bring a new method that eclipses the previous leader. Selman, Kautz, and McAllester (1997) present an excellent summary of the state of the art in propositional reasoning and sketch challenges for coming years. My discussion is therefore brief.

SAT solvers are best distinguished by the type of search they perform: systematic or stochastic.

Systematic SAT Solvers Although it was discovered many years ago, the DPLL algorithm (Davis, Logemann, and Loveland 1962) remains a central algorithm, and it can be summarized with a minimum of background. Let Φ be a CNF formula, that is, a conjunction of clauses (disjunctions). If one of the clauses is just a single literal P , then clearly P must be true to satisfy the conjunction; P is called a unit clause. Furthermore, if some other literal Q exists such that every clause in Φ that refers to Q or $\neg Q$ references Q in the same polarity—for example, all references are true (or all are false)—then Q (or $\neg Q$) is said to be a pure literal. For example, in the CNF formula

$$\begin{aligned} \psi = & (A \vee B \vee \neg E) \wedge (B \vee \neg C \vee D) \wedge (\neg A) \\ & \wedge (B \vee C \vee E) \wedge (\neg D \vee \neg E) \end{aligned}$$

$\neg A$ is a unit clause, and B is a pure literal. I use the notation $\Phi(u)$ to denote the result of setting literal u true and then simplifying. For example, $\Phi(\neg A)$ is

$$\begin{aligned} & (B \vee \neg E) \wedge (B \vee \neg C \vee D) \wedge (B \vee C \vee E) \\ & \wedge (\neg D \vee \neg E) \end{aligned}$$

and $\Phi(B)$ is

$$(\neg A) \wedge (\neg D \vee \neg E)$$

We can now describe DPLL in simple terms; it performs a backtracking, depth-first search through the space of partial truth assignments, using unit-clause and pure-literal heuristics (figure 14). TABLEAU (Crawford and Auton 1993) and SATZ (Li and Anbulagan 1997) are tight implementations of DPLL with careful attention

```

Procedure DPLL(CNF formula:  $\phi$ )
  If  $\phi$  is empty, return yes.
  Else if there is an empty clause in  $\phi$ , return no.
  Else if there is a pure literal  $u$  in  $\phi$ , return DPLL( $\phi(u)$ )
  Else if there is a unit clause  $\{u\}$  in  $\phi$ , return DPLL( $\phi(u)$ )
  Else
    Choose a variable  $v$  mentioned in  $\phi$ .
    If DPLL( $\phi(v)$ ) = yes, then return yes.
    Else return DPLL( $\phi(\neg v)$ ).

```

Figure 14. Backtracking, Depth-First Search through the Space of Partial Truth Assignments.

to data structures and indexing. Many additional heuristics have been proposed to guide the choice of a splitting variable in preparation for the divide-and-conquer recursive call. For example, SATZ selects variables by considering how much unit propagation is facilitated if it branches on the variable (Li and Anbulagan 1997). See Cook and Mitchell (1997) for a discussion of other heuristics.

By incorporating CSP look-back techniques such as conflict-directed back jumping and its generalization, relevance-bounded learning, solver speed was increased substantially (Bayardo and Schrag 1997).

Another interesting direction is the construction of special-purpose SAT solvers, optimized for CNF encodings of planning problems. A first effort in this direction is based on the insight that propositional variables corresponding to action choices are more important than other variables (for example, those corresponding to fluent values) that follow deterministically from action choices. This insight suggests a small change to DPLL: Restrict the choice of splitting variables to action variables. Interestingly, the result of this restriction is dramatic: to four orders of magnitude speedup (Giunchiglia, Massarotto, and Sebastiani 1998). The MODOC solver (Okushi 1998; Van Gelder and Okushi 1998) also uses the high-level structure of the planning problem to speed the SAT solver, but MODOC uses knowledge of which propositions correspond to goals (rather than to actions) to guide its search; the resulting solver is competitive with Walksat (described later).

Stochastic SAT Solvers In contrast to systematic solvers, stochastic methods search locally using random moves to escape from local minima. As a result, stochastic methods are incomplete—when called on hard prob-

lems, a stochastic solver can simply report that it is unable to find a satisfying assignment in the allotted time. This output leaves the observer uninformed because there is no sure way to distinguish an unsatisfiable formula from one whose satisfying assignment is difficult to find. However, stochastic solvers are frequently much faster at finding satisfying assignments when they exist.

The simple and popular GSAT solver is a random-restart, hill-climbing search algorithm (figure 15) (Selman, Levesque, and Mitchell 1992). The successors of a truth assignment are assignments that differ only in the value assigned to a single variable. GSAT performs a greedy search, preferring one assignment over another based on the number of satisfied clauses. Note that the algorithm can move sideways (no change in the number of satisfied clauses) or make negative progress. After hill climbing for a fixed amount of flips (as directed by N_{flips}), GSAT starts anew with a freshly generated, random assignment. After N restarts many of these restarts, GSAT gives up.

WALKSAT (Selman, Kautz, and Cohen 1996, 1994) improves on GSAT by adding additional randomness akin to simulated annealing.¹⁹ On each flip, WALKSAT does one of two things; with probability p , it chooses the same variable GSAT would have chosen; otherwise, it selects a random variable from an unsatisfied clause. Many variants on these algorithms have been constructed and compared, for example, Gent and Walsh (1993) and McAllester, Selman, and Kautz (1997).

An especially promising new method, reported in Gomes, Selman, and Kautz (1998), exploits the fact that the time required by the DPLL procedure is highly dependent on the choice of splitting variable, producing a heavy-tailed distribution of running times (figure 16). They augmented a version of DPLL by (1) adding randomization to the choice of splitting variable and (2) causing the algorithm to quit and restart if it failed to find a solution after a very small time limit t . These restarts curtail unpromising choices (that is, ones that might lead to extremely long running times) before they consume much time. After a number of restarts, the odds are high that the algorithm will stumble on a good choice that leads to a quick solution (that is, one with a running time less than the time limit t). Because very little time is wasted on the restarts, the result is a speedup of several orders of magnitude.

Although stochastic methods can perform extremely well, their performance is usually sensitive to a variety of parameters: random noise p , N_{restarts} , N_{flips} , and so on. Because the

```

Procedure GSAT(CNF formula:  $\phi$ , integer:  $N_{\text{restarts}}$ ,  $N_{\text{flips}}$ )
  For  $i$  equals 1 to  $N_{\text{restarts}}$ ,
    Set  $A$  to a randomly generated truth assignment.
    For  $j$  equals 1 to  $N_{\text{flips}}$ ,
      If  $A$  satisfies  $\phi$ , then return yes.
    Else
      Set  $v$  to be a variable in  $\phi$  whose change gives the largest increase
        in the number of satisfied clauses; break ties randomly.
      Modify  $A$  by flipping the truth assignment of  $v$ .

```

Figure 15. Random-restart, Hill-climbing Search through the Space of Complete Truth Assignments.

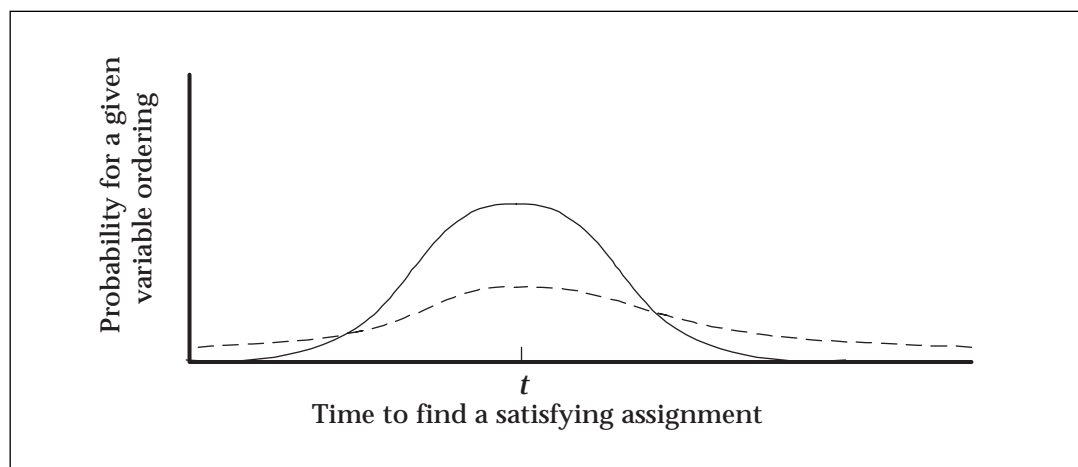


Figure 16. The Time Required by DPLL to Find a Satisfying Assignment Is Highly Dependent on the Order in Which Variables Are Chosen.

Although the distribution of times as a function of variable order varies from problem to problem, many problems show a heavy-tailed distribution (dashed curve) instead of Gaussian (hairline curve). Although t is mean of the Gaussian curve, it is simply the most likely point on the tailed distribution. Indeed, the mean value for a heavy-tailed distribution can be infinite because probability mass stretches rightwards without bound. However, because there is a sizable probability mass to the left of time t , one is likely to land in this area after a small number of restarts sample different orderings.

optimal values for these parameters are a function of the problem being solved and the specific algorithm in question, it can take considerable experimentation to “tune” these parameters for a specific problem distribution. For stochastic methods to reach their potential, automated tuning methods (which don’t require solving complete problem instances!) must be developed; McAllester, Selman, and Kautz (1997) report on work in this direction.

Incremental SAT Solving The problem of propositional satisfiability is closely related to that of truth maintenance (McAllester 1990; de Kleer 1986; Doyle 1979); I focus on LTMS-style truth maintenance systems (McAllester 1980). Both problems concern a CNF formula represented as a set of clauses Σ over a set of propositional variables V . A SAT solver seeks to find a truth assignment (that is, a function

from V to {true; false}) that makes Σ true. An LTMS has two differences: First, an LTMS manipulates a function from V to {true, false, unknown}, which is more general than a truth assignment. Second, an LTMS doesn’t just find this mapping, it maintains it during incremental changes (additions and deletions) to the set of clauses β .

An LTMS uses unit propagation to update its mapping. Any unit clauses can be assigned values immediately, and a clause with a single unknown literal and all remaining literals labeled *false* can also be updated. If a new clause is added to β , it can enable additional inference, and dependency records allow the LTMS to retract inferences that depended on clauses later removed from β . Nayak and Williams (1997) describe an especially efficient method for maintaining this mapping (which

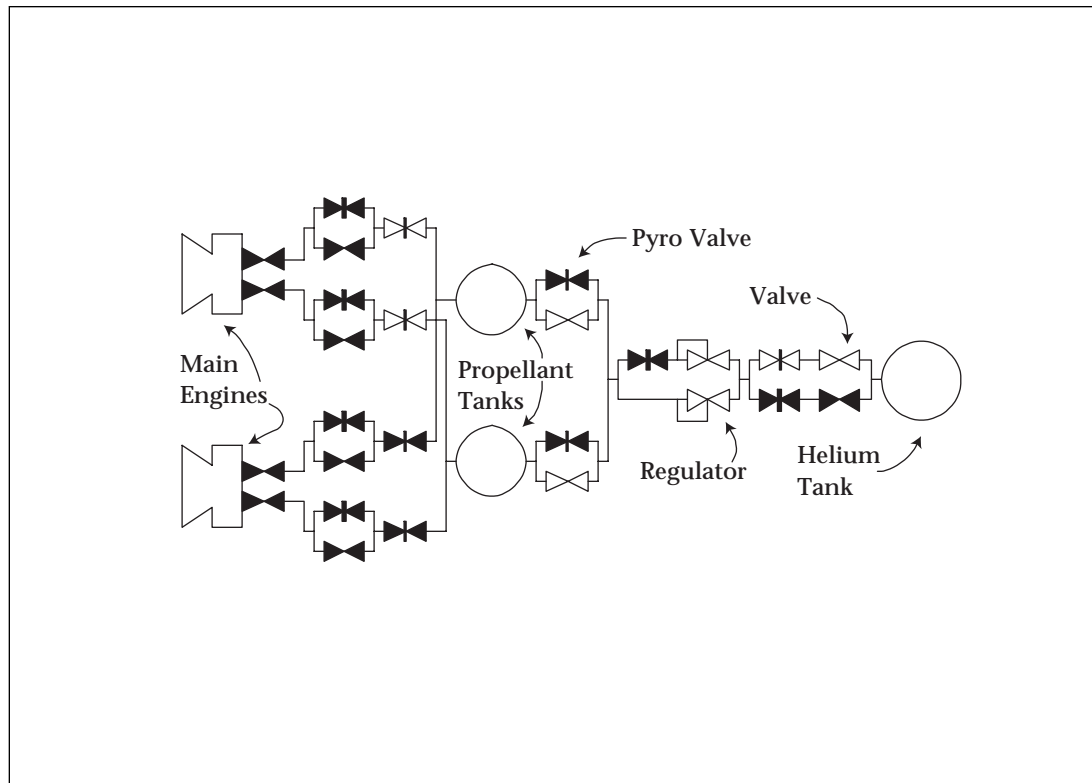


Figure 17. Schematics for Spacecraft Engine (adapted from Williams and Nayak [1997].)

Closed valves are show filled black. Pyro valves can be opened (or closed) only once.

they call an ITMS), and the resulting algorithm is a powerful foundation for building real-time planning and execution systems, as I describe later.

Interleaved Planning and Execution Monitoring

One of the most exciting recent developments is a partially SAT-based reactive control system developed in the context of spacecraft fault protection and configuration management (Williams and Nayak 1997, 1996). Parts of this control system will be demonstrated as part of the remote-agent experiment on NASA's *Deep Space One* mission, which also includes constraint-based temporal planning and scheduling and robust multithreaded execution (Muscettola et al., 1998; Pell et al. 1998, 1997). This reactive control system is a model-based configuration planning and execution system that is best explained with an example. Figure 17 shows a simplified schematic for the main engines of a spacecraft. The helium tank pressurizes the fuel and oxidizer tanks, so that they are forced through valves (if open), to combine in the engines where these propellants ignite to produce thrust. Valves are opened by valve drivers by sending commands through a con-

trol unit. During its long voyage toward its destination (perhaps Saturn), as many components as possible are turned off to save energy, so these control units and drivers must be both turned on and operational before the valves can be adjusted. Radiation makes space a harsh environment that can damage both electronic and physical components. Valves can jam open or shut, and control units can fail in either a soft (resettable) or permanent fashion. To counteract these problems, the engines have a high degree of redundancy (figure 17). However, some of these propellant paths are more flexible than others; for example, pyro valves are less likely to fail but can be switched only once.

The spacecraft's configuration management system must satisfy high-level goals (for example, achieve thrust before orbital insertion) by identifying when failures have occurred and executing actions (for example, powering on control units and switching valves) so that these goals are quickly achieved at minimum cost in terms of power and spent pyro valves. As shown in figure 18, these decisions are made by a pipeline of three major components:

First, the *execution monitor* (MI) interprets limited sensor readings to determine the current physical state of the spacecraft,²⁰ includ-

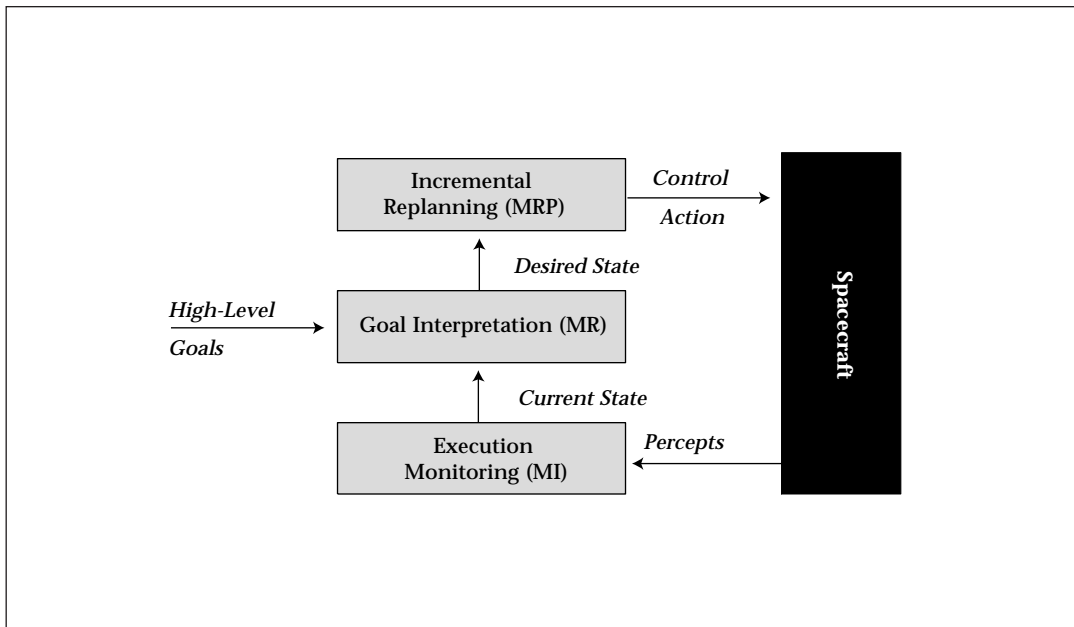


Figure 18. Architecture of the Deep Space One Spacecraft Configuration Planning and Execution System.

ing recognizing when an execution failure has occurred. Frequently, several possible states will be consistent with previous values and current sensors, and in this case, the execution monitor returns the mostly likely single state.

The *goal interpreter* (MR) determines the set of spacecraft states that achieve the high-level goals and are reachable from the current state. It returns the lowest-cost such state, for example, one with minimal power consumption and the fewest blown pyro valves.

The *incremental replanner* (MRP) calculates the first action of a plan to reach the state selected by goal interpretation.

Propositional Encoding of Spacecraft Capabilities

Each of these modules uses a propositional encoding of spacecraft capabilities that (despite superficial differences) is quite similar to the STRIPS domain theories considered earlier in this article. For example, each valve in the engine is described in terms of the following modeling variables: valve-mode, f_{in} , f_{out} , p_{in} , p_{out} . These variables have the domains shown here

valve-mode \in
 {open, closed, stuck-open, stuck-closed}
 f_{in} , $f_{out} \in$ {positive, zero, negative}
 p_{in} , $p_{out} \in$ {high, nominal, low}

Note the use of a discretized, qualitative representation (Weld and de Kleer 1989; Bobrow 1984) of the real-valued flow and pressure vari-

ables. The cross-product of these five variables defines a space of possible valve states, but many of these states are not physically attainable. For example, if the valve is open, there will be equal pressure on both sides, so $p_{in} = p_{out}$. Infeasible states are eliminated by writing a set of propositional logic formulas in which the underlying propositions are of the form, modeling variable = value. For the valve example, one can write²¹

$$\begin{aligned}
 & ((\text{valve-mode} = \text{open}) \vee \\
 & (\text{valve-mode} = \text{stuck-open})) \Rightarrow \\
 & ((p_{in} = p_{out}) \wedge (f_{in} = f_{out})) \\
 & ((\text{valve-mode} = \text{closed}) \vee \\
 & (\text{valve-mode} = \text{stuck-closed})) \Rightarrow \\
 & ((f_{in} = \text{zero}) \wedge (f_{out} = \text{zero}))
 \end{aligned}$$

Note that these descriptions are implicitly parameterized. Each valve has its own variables and, thus, its own propositions. Augmenting these domain axioms, control actions are specified in a temporal, modal logic that adds a “next” operator, O , to propositional logic. For example, the behavior of a valve driver can be described in part with the following formulas:²²

$$\begin{aligned}
 & ((\text{driver-mode} \neq \text{failed}) \wedge (\text{cmd}_{in} = \text{reset})) O \\
 & (\text{driver-mode} = \text{on}) \\
 & ((\text{driver-mode} = \text{on}) \wedge (\text{cmd}_{in} = \text{open}) \wedge \\
 & (\text{valve-mode} \neq \text{stuck-open}) \wedge (\text{valve-mode} \\
 & \neq \text{stuck-closed})) O (\text{valve-mode} = \text{open})
 \end{aligned}$$

Each of these transition equations is akin to a STRIPS operator—the antecedent (that is, left-

hand side) of the implication corresponds to the action name and precondition, and the consequent (right-hand side) equations that follow the \circ are effects that take place in the next time step. The name-parameter distinction is a bit subtle because it stems from the fact that modeling variables are partitioned into disjoint sets: state variables (for example, driver-mode), dependent variables (for example, f_{in}), and control variables (for example, cmd_{in}). The subexpression of the antecedent that consists solely of propositions referring to control variables corresponds to the name of a STRIPS action,²³ and the remainder of the antecedent (that is, propositions referring only to dependent and state variables) corresponds to the STRIPS action precondition. In summary, therefore, Williams and Nayak model the spacecraft with a combination of STRIPS actions and propositional constraints defining the space of feasible states.

Real-Time Inference

Suppose that the agent knows the state of all modeling variables (and, hence, all propositions) at time zero. This information suffices to predict which actions will be executed and, hence, the expected, next spacecraft state. However, actions don't always have their desired effects, so Williams and Nayak included additional transition equations that describe possible failure modes as well. For example, in contrast to the expected result of setting $cmd_{in} = \text{open}$ shown earlier, a failure transition might enumerate the possibility that valve-mode could become stuck-closed. Both normal and failure transition rules are annotated with probabilities. Thus, instead of predicting a unique next spacecraft state, one can predict a ranked list of possible next states, ordered by likelihood.

Given this framework, the processes of execution monitoring and goal interpretation can be cast in terms of combinatorial optimization subject to a set of propositional logic constraints. As input, execution monitoring takes a set of observations of the current values of a subset of the state and dependent variables, and these observations set the values of the corresponding propositions. Thus, execution monitoring is seen to be the problem of finding the most likely next state that is logically consistent with the observations. An incremental SAT solver forms the core of this optimization computation.

Goal interpretation is similar. The objective is to find a spacecraft state that entails the high-level goals and that is most cheaply reached from the state that execution monitor-

ing deems most likely. Estimating the cost of reaching a goal state is relatively easy (for example, one compares the number of switched pyro valves and the differential power use), so logical entailment, computed by the incremental SAT solver, is, again, central.

The incremental replanner takes as input the (most likely) initial state computed by execution monitoring, the (least cost) goal state computed by goal interpretation. As output, the incremental replanner produces an action that is guaranteed to be the first step of a successful,²⁴ cycle-free plan from the initial state to the goal. The beauty of Williams and Nayak's algorithm is its guarantee of a speedy response, which at first glance appears to contradict results showing STRIPS planning is PSPACE complete (Bylander 1991).

Underlying Williams and Nayak's method is the insight that spacecraft configuration planning is far easier than general STRIPS planning because spacecraft engineers specifically designed their creations to be controllable. Williams and Nayak formalize this intuition with a set of crisp constraints that are satisfied by the spacecraft domain. The most important of these restrictions is the presence of a serialization ordering for any (satisfiable) set of goals. As previous theoretical work has shown (Barrett and Weld 1994; Korf 1987), serialized subgoals can be solved extremely quickly because no backtracking is necessary between subgoals. To give an intuitive blocks world example, the set of goals

1. Have block *C* on the table.
2. Have block *B* on block *C*.
3. Have block *A* on block *B*.

is serializable, and solving them in the order 1, 2, 3 is the correct serialization. It doesn't matter how goal 1 is achieved, goals 2 and 3 can be solved without making goal 1 false. Once goals 1 and 2 are achieved, they never need to be violated to solve goal 3. In summary, researchers have long known that serializable goals were an ideal special case, but Williams and Nayak's contribution is twofold: First, they recognized that their spacecraft configuration task was serializable (many real-world domains are not). Second, they developed a fast algorithm for computing the correct order. This last step is crucial because if one attempts to solve a serializable problem in the wrong order, then an exponential amount of time can be wasted by backtracking search. For example, if one solved goal 3 (getting block *A* on block *B*) before solving 1 and 2, the work on goal 3 might be wasted.

Williams and Nayak's goal-ordering algorithm is based on the notion of a *causal*

graph,²⁵ whose vertexes are state variables; a directed edge is present from v_1 to v_2 if a proposition mentioning v_1 is in the antecedent of a transition equation whose consequent mentions v_2 .²⁶ Williams and Nayak observe that the spacecraft-model causal graphs are acyclic, and thus, a topological sort of the graph yields a serialization ordering. If the goals are solved in an upstream order (that is, goals involving v_2 are solved before those of v_1), then no backtracking is required between goals. Essentially all search is eliminated, and the incremental replanner generates control actions in average-case constant time.

Discussion

Although the focus of this article has been on the dramatic explosion in SAT planning and GRAPHPLAN-based algorithms, I close by briefly mentioning some other recent trends.

Planning as Search

Refinement search forms an elegant framework for comparing different planning algorithms and representations (Kambhampati, 1997b; Kambhampati, Knoblock, and Yang 1995). Recent results extend the theory to handle partially HTN domains (Kambhampati, Mali, and Srivastava 1998).

McDermott (1996) showed that an emphasis on (automatically) computing an informative heuristic can make an otherwise simple planner extremely effective. TLPLAN uses (user-provided) domain-specific control information to offset a simple, forward-chaining search strategy—with impressive results (Bacchus and Teh 1998). Geffner demonstrated impressive performance on planning competition problems using heuristic search through the space of world states.

Causal Link Planning

Causal link planners, for example, SNLP (McAllester and Rosenblitt 1991) and UCPOP (Penberthy and Weld 1992), have received less attention in recent years because they are outperformed by GRAPHPLAN and SATPLAN in most domains. However, some of the intuitions underlying these planners have been adopted by the propositional approaches. For example, one of the biggest advantages of causal-link planners, resulting from their backward-chaining regression search, was their insensitivity to irrelevant information in the initial state. Regression focusing (described previously) provides some of these advantages to propositional planners.

One situation where causal link planners

still seem to excel are software domains in which the domain of discourse is unknown to the agent (Etzioni and Weld 1994). When an agent is faced with incomplete information, it cannot construct the Herbrand base and, hence, is unable to use propositional planning methods. Causal-link planners such as XII (Golden, Etzioni, and Weld 1994) and PUC-CINI (Golden 1998), however, work competently.

Handling Uncertainty

Starting with work on the CNLP (Peot and Smith 1992), SENSP (Etzioni et al. 1992), BURIDAN (Kushmerick, Hanks, and Weld 1995, 1994), and C-BURIDAN (Draper, Hanks, and Weld 1994) systems, the AI planning community has more seriously considered extensions to action languages that allow the specification of uncertain effects and incomplete information. Of course, much related work has been performed by the uncertainty in AI community but usually with different assumptions. For example, work on Markov decision processes (MDPs) typically assumes that an agent has complete, immediate, free observability of the world state, even if its own actions are not completely deterministic. Work on partially observable MDPs (POMDPs) relaxes this assumption, but much remains to be done in this area because POMDP solvers are typically much less efficient than MDPs. MDP and POMDP researchers typically state the agent's objective in terms of maximizing a utility function over a fixed, finite horizon. Planning researchers, however, usually seek to achieve a fixed goal configuration, either with complete confidence or with probability greater than some threshold, but no time horizon is considered. In the past, it was thought that planning-based approaches (by their goal-directed natures) were less sensitive to high-dimension state descriptions, that is, the presence of many attributes in the initial state. However, recent work on MDP abstraction and aggregation (Dearden and Boutilier 1997; Boutilier, Dearden, and Goldszmidt 1995) calls this intuition into question. For the field to advance, more work needs to be done comparing these approaches and testing their relative strengths and limitations. Initial results in this area are a start (Littman 1997; Boutilier, Dean, and Hanks 1995), but empirical comparisons are badly needed.

Several researchers have extended GRAPHPLAN to handle uncertainty. *Conformant GRAPHPLAN* (CGP) (Smith and Weld 1998a) handles uncertainty in the initial state and in action effects but does not allow sensing; the resulting conformant plan works in the presence of uncer-

McDermott (1996) showed that an emphasis on (automatically) computing an informative heuristic can make an otherwise simple planner extremely effective.

tainty by choosing robust actions that cover all eventualities. *Sensory GRAPHPLAN* (SGP) (Weld, Anderson, and Smith 1998) extends CGP to allow branching (“contingent”) plans based on run-time information gathered by noiseless sensory actions that might have preconditions. Neither CGP nor SGP incorporates numeric probabilistic reasoning; both build separate planning graph structures for each possible world specified by the problem’s uncertainty, so scaling is a concern. PGRAPHPLAN (Blum and Langford 1998) adopts the MDP framework (that is, numeric probability, complete observability) and builds an optimal n -step, contingent plan using a single planning-graphlike structure to accelerate forward-chaining search (see also Boutilier, Dearden, and Goldszmidt [1995]).

Other researchers have investigated the compilation approach to planning under uncertainty, but instead of compiling to SAT, they target a probabilistic variant called E-MAJSAT:

Given a Boolean formula with *choice variables* (variables whose truth status can be arbitrarily set) and *chance variables* (variables whose truth status is determined by a set of independent probabilities), find the setting of the choice variables that maximizes the probability of a satisfying assignment with respect to the chance variables. (Littman 1997)

Majercik and Littman (1998a) describe a planning compiler based on this idea and present an E-MAJSAT solver akin to DPLL. Caching expensive probability calculations leads to impressive efficiency gains (Majercik and Littman 1998b).

I would be remiss if I failed to mention alternative approaches for handling uncertainty. Interleaved planning and execution (Golden 1998; Golden, Etzioni, and Weld 1994; Ambros-Ingerson and Steel 1988) eschews a contingent plan, instead planning for the expected case and replanning if expectations are violated; although this approach has significant performance advantages, it’s a risky strategy in worlds with irreversible actions where goal failure is costly. Permissive planning (DeJong and Bennett 1997) uses machine learning to bias the planner’s search toward plans that are likely to succeed; the resulting planner is fast because uncertainty is represented only during learning.

Conclusions

In the past few years, the state of the art in AI planning systems has advanced with extraordinary speed. GRAPHPLAN- and SAT-based planning systems can quickly solve problems that

are orders of magnitude harder than those tackled by the best previous planners. Recent developments extend these systems to handle expressive action languages, metric resources, and uncertainty. Type-theoretic domain analysis promises to provide additional speedup, and there are likely more ideas in from the constraint-satisfaction and compiler areas that could usefully be applied. The use of a modern planning system to control a real NASA spacecraft demonstrates that AI planning has matured enough as a field to increase the number of fielded applications. A common thread running through all this research is the use of propositional representations, which support extremely fast inference.

Acknowledgments

I thank Corin Anderson, Mike Ernst, Mark Friedman, Rao Kambhampati, Henry Kautz, Todd Millstein, Bart Selman, David Smith, Brian Williams, and Steve Wolfman for stimulating collaboration and discussion that shaped my understanding of the latest planning techniques. This article has been improved by comments and suggestions from many people, including Jonathan Aldrich, Matthew Cary, Ernest Davis, Jerry DeJong, Nort Fowler, Alfonso Gerevini, Maria Gullickson, Geoff Hulten, Pandu Nayak, Yongshao Ruan, Jude Shavlik, Rein Simmons, Alicen Smith, Vassili Sukharev, Dave Wilkins, and Qiang Yang, but my gratitude to them doesn’t imply that they agree with my perspective on planning. This research was funded by Office of Naval Research grant N00014-98-1-0147, National Science Foundation grant IRI-9303461, and ARPA/Rome Labs grant F30602-95-1-0024.

Notes

1. In particular, I omit any discussion of HTN or mixed-initiative (Ferguson, Allen, and Miller 1996; Ferguson and Allen 1994) planning and have only a brief discussion of topics such as planning under uncertainty, machine-learning approaches, and causal-link planning.
2. The acronym STRIPS stands for Stanford Research Institute problem solver, a famous and influential planner built in the 1970s to control an unstable mobile robot affectionately known as SHAKEY (Fikes and Nilsson 1971).
3. See <ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz> for the PDDL specification.
4. See <ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html> for competition results.
5. Although there is a long history of research applying ideas from constraint satisfaction to planning, I focus on applications to GRAPHPLAN in this article (although compilation of planning to SAT can be viewed as taking the constraint-satisfaction perspective to its logical conclusion). See MOLGEN (Stefik

1981) for seminal work on constraint-posting planning. TWEAK (Chapman 1987), SNLP (McAllester and Rosenblitt 1991), and UCPOP (Penberthy and Weld 1992) manipulated explicit codesignation and ordering constraints. Joslin and Pollack (1996) describe a planner that represented all its decisions as constraints. Kambhampati (1997b) provides a formal framework of planning that compares different planners in terms of the way they handle constraints. GEMPLAN (Lansky 1998) is a modern constraint-posting planner.

6. Similar heuristics have been investigated in the context of causal-link planners; see Smith and Peot (1993); Joslin and Pollack (1994); Yang and Chan (1994); Srinivasan and Howe (1995); Gerevini and Schubert (1996); and Pollack, Joslin, and Paolucci (1997).

7. See Etzioni (1993) and Smith and Peot (1996, 1993) for additional uses of precomputation based on an analysis of action interactions.

8. Fox et al. also point out that their method can dramatically improve the software-engineering process of designing, debugging, and maintaining complex planning domains.

9. My use of the word *only* is too strong because the planning graph might contain some actions that can't ever be executed. Strictly speaking, the planning graph contains a proper superset of the executable actions that is a close approximation of this set.

10. Proof sketch: If A and B appear at both level i and $i - 2$ and are mutex at level i , then by definition, this mutex must be the result of inconsistent effects, interference, or competing needs. If the mutex is the result of the first two reasons, then the mutex will occur at every level containing A and B . However, if the mutex is the result of competing needs, then there are preconditions P and Q of A and B , respectively, such that P is mutex with Q at level $i - 1$. This propositional mutex can only result from the fact that all level $i - 3$ actions supporting P and Q are pairwise mutex, so an inductive argument (combined with action monotonicity) completes the proof.

11. Proof sketch: If they were achievable at level $i - 2$, then adding a level of maintenance actions would achieve them at level i .

12. These approaches make all effects conditional because the action preconditions are added into the antecedent for each conditional effect, and the unavoidable effects (for example, changing the vehicle's location) form a new conditional effect with just the action's preconditions as an antecedent.

13. See the previous subsection entitled Action Schemata, Type Analysis, and Simplification for further explanation of types.

14. It's fine for a given object to have multiple types, but having multiple types must be stated explicitly, or else some form of inheritance reasoning must be added to the graph-expansion process.

15. Note that this definition relies on the fact that type t_1 has a finite universe; as a result, n Skolem constants are generated. If there were two leading, universally quantified variables of the same type, then n^2 Skolem constants ($y_{i,j}$) would be necessary.

16. Note that we are using nonstandard notation here to emphasize the combinatorics. When we write $\text{DriveArg3}(\text{Renton}, t)$ we denote a propositional variable, not a functional term from first-order predicate calculus. Thus, $\text{DriveArg3}(\text{Renton}, t)$ is treated as if it has no substructure. To make this aspect clear, I might better write the symbol DriveArg3Renton_t , but I prefer my notation because it more clearly illustrates the effects of representational differences on CNF size.

17. In fact, the factoring optimization should be applied to all axiom types—not just frame axioms.

18. Contrast my definition of conflict with that of GRAPHPLAN (Blum and Furst 1995) and Kautz and Selman (1996). Unlike Kautz and Selman's parallel encoding, but like their linear one, my encodings have axioms stating that actions imply their effects; their parallel encoding prohibits effect-effect conflicts instead.

19. Download from www.informatik.tu-darmstadt.de/AI/SATLIB.

20. For clarity and consistency, I use different terminology than Williams and Nayak's (1997, 1996) original papers, and I include their acronyms to facilitate correspondence for reader recourse to primary literature. Williams and Nayak name the process of execution monitoring *mode identification*, hence the abbreviation MI. The intuition is that the system's "mode" is its state, and hence, execution monitoring determines whether the system is in the expected state or if a failure has occurred. The process of goal interpretation was called *mode reconfiguration* (hence MR), and what I call incremental replanning was called *model-based reactive planning* (MRP).

21. Note that I am using shorthand here. Because primitive propositions in this representation are of the form modeling variable = value, to encode $p_{in} = p_{out}$, one must expand a formula such as $(p_{in} = \text{positive} \wedge p_{out} = \text{positive}) \vee \dots$

22. Again, I ignore parameterization.

23. By assumption (Williams and Nayak

1997), every transition equation has at least one proposition involving control variables in its antecedent.

24. Here, it is in the absence of failure transitions.

25. It is interesting to compare this work with similar research on subgoal ordering discussed earlier in the subsection entitled Solution Extraction as Constraint Satisfaction. Problem-space graphs (Etzioni 1993) and operator graphs (Smith and Peot 1996, 1993) share many resemblances to causal graphs. Knoblock's (1990) ALPING abstraction system can be viewed as finding a serialization ordering, and it can eliminate most search when given a problem with acyclic structure such as the towers of Hanoi (Knoblock 1992).

26. The causal graph is constructed offline from a compiled version of the domain theory that eliminates all reference to dependent variables.

References

- Ambros-Ingerson, J., and Steel, S. 1988. Integrating Planning, Execution, and Monitoring. In Proceedings of the Seventh National Conference on Artificial Intelligence, 735–740. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Anderson, C.; Smith, D.; and Weld, D. 1998. Conditional Effects in GRAPHPLAN. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, 44–53. Menlo Park, Calif.: AAAI Press.
- Bacchus, F., and Teh, Y. W. 1998. Making Forward-Chaining Relevant. In Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, 54–61. Menlo Park, Calif.: AAAI Press.
- Bacchus, F., and van Run, P. 1995. Dynamic Variable Ordering in CSPs. In *Proceedings of the 1995 Conference on Principles and Practice of Constraint Programming*, 258–275. Heidelberg, Germany: Springer-Verlag.
- Barrett, A., and Weld, D. 1994. Partial Order Planning: Evaluating Possible Efficiency Gains. *Journal of Artificial Intelligence* 67(1): 71–112.
- Bayardo, R., and Schrag, R. 1997. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In Proceedings of the Fourteenth National Conference on Artificial Intelligence, 203–208. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Blum, A., and Furst, M. 1997. Fast Planning through Planning Graph Analysis. *Journal of Artificial Intelligence* 90(1–2): 281–300.
- Blum, A., and Furst, M. 1995. Fast Planning through Planning Graph Analysis. In Pro-

- ceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1636–1642. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Blum, A. L., and Langford, J. C. 1998. Probabilistic Planning in the GRAPHPLAN Framework. In Proceedings of the AIPS98 Workshop on Planning as Combinatorial Search, 8–12. Pittsburgh, Penn.: Carnegie Mellon University.
- Bobrow, D., ed., 1984. *Journal of Artificial Intelligence* (Special Issue on Qualitative Reasoning about Physical Systems) 24(1).
- Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under Uncertainty: Structural Assumptions and Computational Leverage. In Proceedings of the Second European Workshop on Planning, 157–171. Amsterdam, Netherlands: IOS.
- Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting Structure in Policy Construction. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1104–1111. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Bylander, T. 1991. Complexity Results for Planning. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 274–279. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Chapman, D. 1987. Planning for Conjunctive Goals. *Journal of Artificial Intelligence* 32(3): 333–377.
- Cheng, J., and Irani, K. B. 1989. Ordering Problem Subgoals. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 931–936. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Cook, S., and Mitchell, D. 1997. Finding Hard Instances of the Satisfiability Problem: A Survey. In Proceedings of the DIMACS Workshop on Satisfiability Problems, 11–13. Providence, R.I.: American Mathematical Society.
- Crawford, J., and Auton, L. 1993. Experimental Results on the Crossover Point in Satisfiability Problems. In Proceedings of the Eleventh National Conference on Artificial Intelligence, 21–27. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Davis, M.; Logemann, G.; and Loveland, D. 1962. A Machine Program for Theorem Proving. *Communications of the ACM* 5:394–397.
- Dearden, R., and Boutilier, C. 1997. Abstraction and Approximate Decision-Theoretic Planning. *Journal of Artificial Intelligence* 89(1–2): 219–283.
- DeJong, G., and Bennett, S. 1997. Permissive Planning: Extending Classical Planning to Uncertain Task Domains. *Journal of Artificial Intelligence* 89(1–2): 173–217.
- de Kleer, J. 1986. An Assumption-Based Truth Maintenance System. *Journal of Artificial Intelligence* 28(2): 127–162.
- Doyle, J. 1979. A Truth Maintenance System. *Journal of Artificial Intelligence* 12(3): 231–272.
- Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic Planning with Information Gathering and Contingent Execution. In Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, 31–36. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Ernst, M.; Millstein, T.; and Weld, D. 1997. Automatic SAT-Compilation of Planning Problems. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, 1169–1176. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Erol, K.; Hendler, J.; and Nau, D. 1994. HTN Planning: Complexity and Expressivity. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 1123–1128. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Etzioni, O. 1993. Acquiring Search-Control Knowledge via Static Analysis. *Artificial Intelligence* 62(2): 255–302.
- Etzioni, O., and Weld, D. 1994. A Softbot-Based Interface to the Internet. *Communications of the ACM* 37(7): 72–6.
- Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An Approach to Planning with Incomplete Information. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 115–125. San Francisco, Calif.: Morgan Kaufmann.
- Falkenhainer, B., and Forbus, K. 1988. Setting Up Large Scale Qualitative Models. In Proceedings of the Seventh National Conference on Artificial Intelligence, 301–306. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Ferguson, G., and Allen, J. 1994. Arguing about Plans: Plan Representation and Reasoning in Mixed-Initiative Planning. In Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, 43–48. Menlo Park, Calif.: AAAI Press.
- Ferguson, G.; Allen, J.; and Miller, B. 1996. TRAINS-95: Towards a Mixed-Initiative Planning Assistant. In Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, 70–77. Menlo Park, Calif.: AAAI Press.
- Fikes, R., and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Journal of Artificial Intelligence* 2(3–4): 189–208.
- Fox, M., and Long, D. 1998. The Automatic Inference of State Invariants in TIM. Technical Report 11/98, University of Durham.
- Gazen, B., and Knoblock, C. 1997. Combining the Expressivity of UCPOP with the Efficiency of GRAPHPLAN. In *Proceedings of the Fourth European Conference on Planning*, 221–233. Berlin: Springer-Verlag.
- Genesereth, M., and Nilsson, N. 1987. *Logical Foundations of Artificial Intelligence*. San Francisco, Calif.: Morgan Kaufmann.
- Gent, I., and Walsh, T. 1993. Towards an Understanding of Hill-Climbing Procedures for SAT. In Proceedings of the Eleventh National Conference on Artificial Intelligence, 28–33. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Gerevini, A., and Schubert, L. 1998. Inferring State Constraints for Domain-Independent Planning. In Proceedings of the Fifteenth National Conference on Artificial Intelligence, 905–912. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Gerevini, A., and Schubert, L. 1996. Accelerating Partial-Order Planners: Some Techniques for Effective Search Control and Pruning. *Journal of Artificial Intelligence Research* 5:95–137.
- Giunchiglia, E.; Massarotto, A.; and Sebastiani, R. 1998. Act, and the Rest Will Follow: Exploiting Determinism in Planning as Satisfiability. In Proceedings of the Fifteenth National Conference on Artificial Intelligence, 948–953. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Golden, K. 1998. Leap before You Look: Information Gathering in the PUCINI Planner. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, 70–77. Menlo Park, Calif.: AAAI Press.
- Golden, K.; Etzioni, O.; and Weld, D. 1994. Omnipotence without Omniscience: Sensor Management in Planning. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 1048–1054. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Gomes, C.; Selman, B.; and Kautz, H. 1998. Boosting Combinatorial Search through Randomization. In Proceedings of the Fifteenth National Conference on Artificial Intelligence, 431–437. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Green, C. 1969. Application of Theorem

- Proving to Problem Solving. In Proceedings of the First International Joint Conference on Artificial Intelligence, 219–239. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Haas, A. 1987. The Case for Domain-Specific Frame Axioms. In *The Frame Problem in Artificial Intelligence, Proceedings of the 1987 Workshop*, 115–128. San Francisco, Calif.: Morgan Kaufmann.
- Haralick, R. M., and Elliott, G. L. 1980. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Journal of Artificial Intelligence* 14(3): 263–313.
- Irani, K. B., and Cheng, J. 1987. Subgoal Ordering and Goal Augmentation for Heuristic Problem Solving. In Proceedings of the Tenth International Joint Conference on Artificial Intelligence, 1018–1024. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Joslin, D., and Pollack, M. 1996. Is “Early Commitment” in Plan Generation Ever a Good Idea? In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1188–1193. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Joslin, D., and Pollack, M. 1994. Least-Cost Flaw Repair: A Plan Refinement Strategy for Partial-Order Planning. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 1004–1009. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Kambhampati, S. 1998a. EBL and DDB for GRAPHPLAN. TR-99-008, Department of Computer Science and Engineering, Arizona State University.
- Kambhampati, S. 1998b. On the Relations between Intelligent Backtracking and Failure-Driven Explanation-Based Learning in Constraint Satisfaction and Planning. TR-97-018, Department of Computer Science and Engineering, Arizona State University.
- Kambhampati, S. 1997a. Challenges in Bridging Plan Synthesis Paradigms. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, 44–49. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Kambhampati, S. 1997b. Refinement Planning as a Unifying Framework for Plan Synthesis. *AI Magazine* 18(2): 67–97.
- Kambhampati, S.; Knoblock, C.; and Yang, Q. 1995. Planning as Refinement Search: A Unified Framework for Evaluating Design Trade-Offs in Partial Order Planning. *Journal of Artificial Intelligence* 76(1–2): 167–238.
- Kambhampati, R.; Lambrecht, E.; and Parker, E. 1997. Understanding and Extending GRAPHPLAN. In *Proceedings of the Fourth European Conference on Planning*, 260–272. Berlin: Springer-Verlag.
- Kambhampati, R.; Mali, A.; and Srivastava, B. 1998. Hybrid Planning for Partially Hierarchical Domains. In Proceedings of the Fifteenth National Conference on Artificial Intelligence, 882–888. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Kautz, H., and Selman, B. 1998a. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. In AIPS98 Workshop on Planning as Combinatorial Search, 58–60. Pittsburgh, Penn.: Carnegie Mellon University.
- Kautz, H., and Selman, B. 1998b. The Role of Domain-Specific Knowledge in the Planning as Satisfiability Framework. In Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, 181–189. Menlo Park, Calif.: AAAI Press.
- Kautz, H., and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1194–1201. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, 359–363. Chichester, U.K.: Wiley.
- Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding Plans in Propositional Logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, 374–384. San Francisco, Calif.: Morgan Kaufmann.
- Kelleher, K., and Cohen, P. 1992. Automatically Synthesizing Domain Constraints from Operator Descriptions. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, 653–655. Chichester, U.K.: Wiley.
- Knoblock, C. 1995. Planning, Executing, Sensing, and Replanning for Information Gathering. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1686–1693. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Knoblock, C. 1992. An Analysis of ABSTRIPS. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, 126–135. San Francisco, Calif.: Morgan Kaufmann.
- Knoblock, C. 1991. Automatically Generating Abstractions for Problem Solving. Technical report, CMU-CS-91-120, Ph.D. diss., Carnegie Mellon University.
- Knoblock, C. 1990. Learning Abstraction Hierarchies for Problem Solving. In Proceedings of the Eighth National Conference on Artificial Intelligence, 923–928. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Koehler, J. 1998a. Planning under Resource Constraints. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, 489–493. Chichester, U.K.: Wiley.
- Koehler, J. 1998b. Solving Complex Planning Tasks through Extraction of Subproblems. In Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, 62–69. Menlo Park, Calif.: AAAI Press.
- Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997a. Extending Planning Graphs to an ADL Subset. In *Proceedings of the Fourth European Conference on Planning*, 273–285. Berlin: Springer-Verlag.
- Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997b. Extending Planning Graphs to an ADL Subset. TR 88, Institute for Computer Science, University of Freiburg.
- Kondrack, G., and van Beek, P. 1997. A Theoretical Evaluation of Selected Backtracking Algorithms. *Journal of Artificial Intelligence* 89(1-2): 365–387.
- Korf, R. 1987. Planning as Search: A Quantitative Approach. *Journal of Artificial Intelligence* 33(1): 65–88.
- Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An Algorithm for Probabilistic Planning. *Journal of Artificial Intelligence* 76(1–2): 239–286.
- Kushmerick, N.; Hanks, S.; and Weld, D. 1994. An Algorithm for Probabilistic Least Commitment Planning. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 1073–1078. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Lansky, A. L. 1998. Localized Planning with Action-Based Constraints. *Journal of Artificial Intelligence* 98(1–2): 49–136.
- Li, C., and Anbulagan. 1997. Heuristics Based on Unit Propagation for Satisfiability Problems. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, 366–371. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Littman, M. 1997. Probabilistic Propositional Planning: Representations and Complexity. In Proceedings of the Fourteenth National Conference on Artificial Intelligence, 748–754. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Long, D., and Fox, M. 1998. Efficient Implementation of the PLAN GRAPH in STAN. Technical Report, TR 14/98, Durham University.
- McAllester, D. 1990. Truth Maintenance. In Proceedings of the Eighth National Conference on Artificial Intelligence, 1109–1116. Menlo Park, Calif.: American Association

for Artificial Intelligence.

McAllester, D. 1980. An Outlook on Truth Maintenance. AI memo, 551, AI Lab, Massachusetts Institute of Technology.

McAllester, D., and Rosenblitt, D. 1991. Systematic Nonlinear Planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 634–639. Menlo Park, Calif.: American Association for Artificial Intelligence.

McAllester, D.; Selman, B.; and Kautz, H. 1997. Evidence for Invariants in Local Search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 321–326. Menlo Park, Calif.: American Association for Artificial Intelligence.

McCarthy, J., and Hayes, P. J. 1969. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence 4*, 463–502. Edinburgh, U.K.: Edinburgh University Press.

McDermott, D. 1996. A Heuristic Estimator for Means-Ends Analysis in Planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, 142–149. Menlo Park, Calif.: AAAI Press.

Majercik, S. M., and Littman, M. L. 1998a. MAX-PLAN: A New Approach to Probabilistic Planning. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, 86–93. Menlo Park, Calif.: AAAI Press.

Majercik, S. M., and Littman, M. L. 1998b. Using Caching to Solve Larger Probabilistic Planning Problems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 954–960. Menlo Park, Calif.: American Association for Artificial Intelligence.

Mali, A. D., and Kambhampati, S. 1998. Encoding HTN Planning in Propositional Logic. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, 190–198. Menlo Park, Calif.: AAAI Press.

Minton, S.; Carbonell, J. G.; Knoblock, C. A.; Kuokka, D. R.; Etzioni, O.; and Gil, Y. 1989. Explanation-Based Learning: A Problem-Solving Perspective. *Journal of Artificial Intelligence* 40(1–3): 63–118.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence* 103(1-2): 5–48.

Nayak, P., and Williams, B. 1997. Fast Context Switching in Real-Time Propositional Reasoning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 50–56. Menlo Park, Calif.: American Association for Artificial Intelligence.

Nebel, B.; Dimopoulos, Y.; and Koehler, J.

1997. Ignoring Irrelevant Facts and Operators in Plan Generation. In *Proceedings of the Fourth European Conference on Planning*, 338–350. Berlin: Springer-Verlag.

Okushi, F. 1998. Parallel Cooperative Propositional Theorem Proving. In *Proceedings of the Fifth International Symposium on Artificial Intelligence and Mathematics*. Dordrecht, The Netherlands: Kluwer Academic.

Pednault, E. 1989. ADL: Exploring the Middle Ground between STRIPS and the Situation Calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 324–332. San Francisco, Calif.: Morgan Kaufmann.

Pell, B.; Bernard, D.; Chien, S.; Gat, E.; Muscettola, N.; Nayak, P.; Wagner, M.; and Williams, B. 1997. An Autonomous Spacecraft Agent Prototype. In *Proceedings of the First International Conference on Autonomous Agents*, 253–261. New York: Association for Computing Machinery.

Pell, B.; Bernard, D. E.; Chien, S. A.; Gat, E.; Muscettola, N.; Nayak, P. P.; Wagner, M. D.; and Williams, B. C. 1998. An Autonomous Spacecraft Agent Prototype. *Autonomous Robots* 5(1): 93–108.

Penberthy, J., and Weld, D. 1992. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 103–114. San Francisco, Calif.: Morgan Kaufmann.

Peot, M., and Smith, D. 1992. Conditional Nonlinear Planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, 189–197. San Francisco, Calif.: Morgan Kaufmann.

Pollack, M. E.; Joslin, D.; and Paolucci, M. 1997. Flaw Selection Strategies for Partial-Order Planning. *Journal of Artificial Intelligence Research* 6:223–262.

Rintanen, J. T. 1998. A Planning Algorithm Not Based on Directional Search. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, 953–960. San Francisco, Calif.: Morgan Kaufmann.

Selman, B.; Kautz, H.; and Cohen, B. 1996. Local Search Strategies for Satisfiability Testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 26:521–532.

Selman, B.; Kautz, H.; and Cohen, B. 1994. Noise Strategies for Improving Local Search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 337–343. Menlo Park, Calif.: American Association for Artificial Intelligence.

Selman, B.; Kautz, H.; and McAllester, D. 1997. Computational Challenges in Propositional Reasoning and Search. In *Proceed-*

ings of the Fifteenth International Joint Conference on Artificial Intelligence, 50–54. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 440–446. Menlo Park, Calif.: American Association for Artificial Intelligence.

Smith, D. 1989. Controlling Backward Inference. *Journal of Artificial Intelligence* 39(2): 145–208.

Smith, D., and Peot, M. 1996. Suspending Recursion in Causal Link Planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, 182–189. Menlo Park, Calif.: AAAI Press.

Smith, D., and Peot, M. 1993. Postponing Threats in Partial-Order Planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 500–506. Menlo Park, Calif.: American Association for Artificial Intelligence.

Smith, D., and Weld, D. 1998a. Conformant GRAPHPLAN. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 889–896. Menlo Park, Calif.: American Association for Artificial Intelligence.

Smith, D., and Weld, D. 1998b. Temporal GRAPHPLAN. Technical report, 98-09-06, Department of Computer Science and Engineering, University of Washington.

Srinivasan, R., and Howe, A. 1995. Comparison of Methods for Improving Search Efficiency in a Partial-Order Planner. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1620–1626. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Stefik, M. 1981. Planning with Constraints (MOLGEN: Part 1). *Journal of Artificial Intelligence* 14(2): 111–139.

Tenenberg, J. 1988. Abstraction in Planning. Ph.D. thesis, Department of Computer Science, University of Rochester.

Van Gelder, A., and Okushi, F. 1998. A Propositional Theorem Prover to Solve Planning and Other Problems. In *Proceedings of the Fifth International Symposium on Artificial Intelligence and Mathematics*. Dordrecht, The Netherlands: Kluwer Academic.

Van Gelder, A., and Tsuji, Y. K. 1996. Satisfiability Testing with More Reasoning and Less Guessing. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, eds. D. S. Johnson and M. Trick, 559–586. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Providence, R.I.: American Mathe-

mathematical Society.

Veloso, M. 1994. Flexible Strategy Learning: Analogical Replay of Problem-Solving Episodes. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 595–600. Menlo Park, Calif.: American Association for Artificial Intelligence.

Weld, D. 1994. An Introduction to Least Commitment Planning. *AI Magazine* 15(4): 27–61.

Weld, D., and de Kleer, J., eds. 1989. *Readings in Qualitative Reasoning about Physical Systems*. San Francisco, Calif.: Morgan Kaufmann.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending GRAPHPLAN to Handle Uncertainty and Sensing Actions. In Proceedings of the Fifteenth National Conference on Artificial Intelligence, 897–904. Menlo Park, Calif.: American Association for Artificial Intelligence.

Williams, B. C., and Nayak, P. P. 1997. A Reactive Planner for a Model-Based Execution. In Proceedings of the Fifteenth International Joint Conference on Artificial

Intelligence, 1178–1185. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Williams, B. C., and Nayak, P. P. 1996. A Model-Based Approach to Reactive Self-Configuring Systems. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 971–978. Menlo Park, Calif.: American Association for Artificial Intelligence.

Yang, Q. 1990. Formalizing Planning Knowledge for Hierarchical Planning. *Computational Intelligence* 6(1): 12–24.

Yang, Q., and Chan, A. 1994. Delaying Variable Binding Commitments in Planning. In Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, 182–187. Menlo Park, Calif.: AAAI Press.

Yang, Q., and Tenenber, J. 1990. ABTWEAK: Abstracting a Nonlinear, Least Commitment Planner. In Proceedings of the Eighth National Conference on Artificial Intelligence, 204–209. Menlo Park, Calif.: American Association for Artificial Intelligence.

Daniel Weld received bachelor's degrees in both computer science and biochemistry at Yale University in 1982. He received a Ph.D. from the Massachusetts Institute of Technology Artificial Intelligence Lab in 1988 and immediately joined the Department of Computer Science and Engineering at the University of Washington, where he is now a professor. Weld received a Presidential Young Investigator's award in 1989 and an Office of Naval Research Young Investigator's award in 1990 and is a fellow of the American Association for Artificial Intelligence. Weld is on the advisory board of the *Journal of Artificial Intelligence Research*, has been guest editor for *Computational Intelligence and Artificial Intelligence*, and was program chair for the 1996 National Conference on Artificial Intelligence. Weld founded Netbot Inc., which developed the JANGO comparison shopping agent (now part of the Excite Shopping Channel), and AdRelevance, Inc., which is transforming internet advertising. Weld has published about 100 technical papers on AI, planning, data integration, and software agents.