Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search

Henry Kautz and **Bart Selman**
AT&T Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
{kautz, selman}@research.att.com
http://www.research.att.com/~{kautz, selman}

Abstract

Planning is a notoriously hard combinatorial search problem. In many interesting domains, current planning algorithms fail to scale up gracefully. By combining a general, stochastic search algorithm and appropriate problem encodings based on propositional logic, we are able to solve hard planning problems many times faster than the best current planning systems. Although stochastic methods have been shown to be very effective on a wide range of scheduling problems, this is the first demonstration of its power on truly challenging classical planning instances. This work also provides a new perspective on representational issues in planning.

Introduction

There is a widespread belief in the AI community that planning is not amenable to general theorem-proving techniques. The origin of this belief can be traced to the early 1970's, when work on plan generation using first-order, resolution theorem-proving (Green 1969) failed to scale up to realistically-sized problems. The relative success of the STRIPS system (Fikes and Nilsson 1971) established the basic paradigm for practically all subsequent work in planning. Planning is viewed as a systematic search through either a state-space or through a space of partial plans. Different representations are used for actions and for states or fluents. Control strategies are not discussed in terms of general rules of inference, but rather in terms of rules for establishing and protecting goals, detecting conflicts between actions, and so forth.

The results described in this paper challenge this belief. We have applied general reasoning systems to the task of plan synthesis, and obtained results that are competitive with, and in many cases superior to, the best specialized planning systems. Why was this possible? We believe that the lesson of the 1970's should not have been that planning required specialized algorithms, but simply that *first-order deductive* theorem-proving does not scale well. By contrast, the past few years have seen dramatic progress in the size of problems that can be handled by *propositional satisfiability* testing programs (Trick and Johnson 1993, Selman 1995). In particular, new algorithms based on randomized local search (Selman *et al.* 1992) can solve certain classes of hard problems that are an order of magnitude larger than those that can be solved by older approaches. Therefore, our formalization of planning is based on propositional satisfiability, rather than first-order refutation.

We ran experiments with both one of the best systematic satisfiability algorithms ("tableau", by Crawford and Auton (1993)) and one of the best stochastic algorithms ("Walksat", by Selman *et al.* (1994; 1996)). All task-specific information was given a uniform clausal representation: the inference engines had no explicit indication as to what stood for a goal or what stood for an operator. This meant that the solvers were not constrained to perform a strict backward or forward chaining search, as would be done by most planning systems. Far from being a disadvantage, this greatly adds to the power of approach, by allowing constraints to propagate more freely and thus more quickly reduce the search space. (The idea of viewing planning as general constraint satisfaction rather than directional search has also been explored by other authors; see, for example, Joslin and Pollack (1995).)

The notion of formalizing planning as *propositional* reasoning immediately raises certain questions. Planning is a notoriously hard problem. In fact, the general plan-existence problem for STRIPS-style operators is PSPACE-complete (Bylander 1991, Erol *et al.* 1992, Backstrom 1992). How, then, is it possible to formulate planning as only an NP-complete problem? This difficulty disappears when we realize that the PSPACE-hardness result only holds when the potential solutions can be of exponential length. If we are only interested in polynomial-length plans, then planning is indeed NP-complete.

Many other planning systems can be viewed as specialized propositional reasoning engines. A surprisingly efficient recent planning system is Graphplan, developed by Blum and Furst (1995). Graphplan works in two phases: in the first, a problem stated using STRIPS notation is converted to a data structure called a "planning graph". In the second, the graph is systematically searched for a solution. The planning graph is in fact a propositional representation. In some of our experiments, we directly converted planning graphs into sets of clauses, and then applied Walksat or tableau. For other experiments we developed by hand even more compact and efficient clausal encodings of the problems. As we will see, it was often the case that Walksat dramatically outperformed both the general and specialized systematic search engines.

The success of stochastic local search for planning may come as a surprise. Although local search has been successfully applied to *scheduling* problems (Adorf and Johnston 1990, Minton *et al.* 1990, 1992), it has seen little use for planning. Some authors (Kautz and Selman (1992), Crawford and Baker (1994)) have suggested that planning (finding a partially-ordered set of operators that achieve a goal) and scheduling (assigning times and resources to a given, fixed set of operators) require different control mechanisms, and that planning is inherently a systematic process. Our present success can be mainly attributed to two factors: first, the greater speed and power of Walksat over earlier local search satisfiability algorithms (*e.g.*, GSAT (Selman *et al.* 1992)); and second, our use of better problem encodings – including "compiling away" plan operators, and extending a technique from Blum and Furst for encoding partially-ordered plans with parallel actions. Our results appear to be the first convincing evidence that stochastic local search is indeed a powerful technique for planning.

We will discuss techniques for encoding planning problems as propositional SAT in some detail below. Our experience has been that the search for domain axiomatizations with better computational properties has led us to valuable insights at the representational level. For example, we will describe one encoding we used that "compiles away" any explicit propositions that stand for actions, leaving only fluents. While this encoding was initially motivated by a concern for reducing the number of different propositions in the final formula, it turned out to also enable a particularly simple and elegant solution to the frame problem with parallel actions. It is important to note that we are emphatically *not* suggesting that control knowledge should be "mixed-in" with declarative information, as occurs in logic programming. Instead, we are suggesting that it can be advantageous to try to optimize the *gross statistical properties* of an axiomatization when developing or choosing between declarative representations.

This paper is organized as follows. After a short preview of the results, we discuss general approaches to planning as satisfiability, and particular encoding techniques. We then present experimental results drawn from several domains, including logistics problems, the "rocket" domain, and the blocks world. We compare the performance of both systematic and stochastic algorithms on different kinds of SAT encodings to the performance of Graphplan, and cite comparisons of Graphplan with the well-known Prodigy (Carbonell *et al.* 1992, Stone *et al.* 1994) and UCPOP (Penberthy and Weld 1992) systems.

Preview of Results

Before we describe our approach in detail, we will first highlight some of our main experimental results. In order to evaluate our method, we considered planning domains that lead to serious computational difficulties in traditional planners. Barrett and Weld (1994) discuss various characteristics of such domains. In general, the hardest planning domains contain intricate interactions between planning operators, and various types of goal and subgoal interactions. These interactions complicate the order in which the goals and subgoals should be established, and make it difficult to select the right operator for establishing a goal. Real-world domains often contain both sources of computational difficulties.

In our experiments, we focussed on two natural domains: the "rocket" domain (Blum and Furst 1995) and the "logistics" domain (Veloso 1992). Blum and Furst showed that Graphplan outperforms Prodigy and UCPOP on the rocket problems. We extended this problem somewhat to make it more challenging for Graphplan. The logistics domain can be viewed as yet a further extension of the rocket domain, making it even harder. We also considered several relatively large blocks world problems, because even *small* blocks world instances are often already surprisingly hard for traditional planners.

Table 1 gives the results on some of the hardest instances we considered.[1] From the last column, it is clear that using a stochastic method (Walksat) and a direct SAT encoding, we can solve these instances two or more orders of magnitude faster than Graphplan. Walksat actually found the optimal (*i.e.*, shortest possible plans) for these problems. Thus, for example, it found an optimal 36 step plan to the blocks world problem "bw_large.d". This instance contains 19 blocks and has multiple stacks in both the initial and goal state. We not aware of any other planning algorithm that can solve instances this size without incorporating domain-specific search control knowledge.

The table also contains our results of running on the SAT encodings *derived* from Graphplan's planning graphs. The first two instances are again solved significantly faster than by using Graphplan itself. The SAT encodings for the last two instances became too large for our SAT procedures. Our SAT encoding is more compact than the original Graphplan representation, but Graphplan can handle larger internal data structures than can our SAT procedures. The planning graph for "bw_large.b" contains 18,069 nodes and has over one million exclusion relations. This is just within Graphplan's reach, taking over 7 hours; on our state-base encoding, Walksat takes only 22 seconds. Walksat's solution was proved optimal using the systematic algorithm tableau. Interestingly, although tableau is able to show that there is no shorter solution, it cannot actually find the solution itself! This show how stochastic and the systematic methods can complement one another.

Planning as Satisfiability

While planning has traditionally been formulated as deduction in first-order logic (Green 1969, McCarthy and Hayes 1969, Pednault 1988, Allen 1991), Kautz

[1] Walksat and Graphplan are implemented in C and ran on an SGI Challenge with a 150 MHz MIPS R4400 processor. We thank Avrim Blum for providing us with his code. Tableau ran on a SPARC-10 processor. We thank Jimi Crawford for making a sparc executable of the latest version of his tableau program available to us. The tableau code is optimized for the sparc architecture. Based on previous code comparisons, we estimate the tableau code would run approximately 20% faster on our SGI processor. Our experimental data and code is available from the authors.

problem	time / actions	Graphplan	SAT from planning graph		Direct SAT encoding
			systematic	stochastic	stochastic
rocket_ext.a	7/34	520	4.4	4.7	0.1
logistics.c	13/65	—	23,040	240	1.9
bw_large.b	9/18	27,115	—	—	22
bw_large.d	18/36	—	—	—	937

Table 1: Preview of experimental results. Times in seconds. A long dash (—) indicates that the experiment was terminated after 10 hrs with no solution found.

and Selman (1992) formalized planning in terms of propositional satisfiability. In this framework, a plan corresponds to any model (*i.e.*, truth-assignment) that satisfies a set of logical constraints that represent the initial state, the goal state, and domain axioms. Time consists of a fixed, discrete number of instances. A proposition corresponds either to a time-varying condition (a *fluent*) holding at a particular instant (*e.g.*, `on(A,B,3)`), or to an action that begins to occur at the specified instance and ends at the following instance (*e.g.*, `pickup(A,3)`). General constraints over facts and actions are written as axiom schemas, which are then instantiated for the objects and number of time instances used by a particular problem. The maximal length of a plan is thus fixed at instantiation time; if this quantity is not known in advance, it is straightforward to perform a binary search on instantiations of various sizes, to find the smallest for which a solution is found. (For example, if the optimal plan length is 7, the search would proceed through plans of length 2, 4, 8 (plan found), 6 (no plan found), and finally 7.)

The satisfiability approach can be directly implemented using SAT algorithms, that in general have much better scaling properties than deductive FOL theorem provers. Another advantage is its expressive power. It is easy to represent arbitrary constraints over intermediate states (not just the initial and goal states), and over the structure of the plan itself. For example, to assert that every `pickup` is immediately followed by a `stack`, one could write a schema like

$$\texttt{pickup(x,i)} \supset \exists \texttt{y.stack(x,y,i+1)}.$$

It is quite hard to represent these kinds of constraints in STRIPS. Finally, because it a "real logic" as opposed to STRIPS, the relationships between predicates can be stated explicitly, and it is unnecessary to distinguish "primitive" from "derived" predicates. For example, most STRIPS-style operators for planning handle the predicates `clear` and `on` separately, whereas in our framework one can simply assert

$$\texttt{clear(x,i)} \equiv \neg\exists \texttt{y.on(y,x,i)}$$

The domain axioms for the satisfiability approach are in general stronger than those used by the deductive framework, because it is necessary to rule out all "unintended" models. We will describe several ways this can be done: *(i)* encodings derived from the planning graphs of Graphplan (Blum and Furst 1995); *(ii)* the linear encodings of Kautz and Selman (1992); and *(iii)* general state-based encodings, which incorporate the best features of the previous two. We refer to the both the linear and state-based encodings as "direct" encodings.

Graphplan-based Encodings

As mentioned above, the Graphplan system (Blum and Furst 1995) works by converting a STRIPS-style specification into a planning graph. This is an ordered graph, where alternating layers of nodes correspond to grounds facts (indexed by the time step for that layer) and fully-instantiated operators (again indexed by the time step). Arcs lead from each fact to the operators that contain it as a precondition in the next layer, and similarly from each operator to its effects in the next layer. For every operator layer and every fact there is also a no-op "maintain" operator that simply has that fact as both a precondition and "add" effect.

A solution is a subgraph of the planning graph that contains all the facts in the initial and goal layers, and contains no two operators in the same layer that conflict (*i.e.*, one operator deletes a precondition or an effect of the other). Thus, a solution corresponds to a partially-ordered plan, which may contain several operators occuring at the same time step, with the semantics that those operators may occur in any order (or even in parallel). For planning problems that can take advantage of this kind of parallelism, the planning graph can have many fewer layers than the number of steps in a linear solution — and therefore be much smaller.

A planning graph is quite similar to a propositional formula, and in fact, we were able to automatically convert planning graphs into CNF notation. The translation begins at goal-layer of the graph, and works backward. Using the "rocket" problem in Blum and Furst (1995, Fig. 2) as an example (where "`load(A,R,L,i)`" means "load `A` into `R` at location `L` at time `i`", and "`move(R,L,P,i)`" means "move `R` from `L` to `P` at time `i`"), the translation is:

- the initial state holds at layer 1, and the goals hold at the highest layer;
- each fact at level i implies the disjunction of all the operators at level $i-1$ that have it as an add-effect; *e.g.*,

  $$\texttt{in(A,R,3)} \supset (\texttt{load(A,R,L,2)} \vee \texttt{load(A,R,P,2)} \vee \texttt{maintain(in(A,R),2)})$$

- operators imply their preconditions, *e.g.*,

  $$\texttt{load(A,R,L,2)} \supset (\texttt{at(A,L,1)} \wedge \texttt{at(R,L,1)})$$

- conflicting actions are mutually exclusive; *e.g.*,

  $$\neg\texttt{load(A,R,L,2)} \vee \neg\texttt{move(R,L,P,2)}$$

Graphplan uses a set of rules to propagate the effects of mutually exclusive actions, leading to additional exclusiveness constraints. In our logical formulation, these additional constraints are logically implied by the original formulation.

Linear Encodings

Kautz and Selman (1992) described a set of sufficient conditions for ensuring that all models of the domain axioms, initial, and goal states correspond to valid plans. These were:

- an action implies *both* its preconditions and effects;
- exactly one action occurs at each time instant;
- the initial state is completely specified;
- classical frame conditions for all actions (*i.e.*, if an action does not change the truth condition of fact, then the fact remains true or remains false when the action occurs).

Intuitively, the first condition makes sure that actions only occur when their preconditions hold, and the "single action" and frame axioms force any state that follows a legal state to also be a legal state. Models under this encoding correspond to linear plans; as the number of operators in a plan increases, these encodings become very large. Kautz and Selman observed that the number of propositional variables can be significantly reduced by replacing certain predicates that take two or more arguments (plus a time-index argument) with ones that take a single argument (plus a time-index). For example, instead of the predicate `move(x,y,z,i)` (meaning "move block `x` from `y` to `z` at time `i`"), they used three predicates, `object`, `source`, and `destination`, with the correspondence

$$\texttt{move(x,y,z,i)} \equiv (\texttt{object(x,i)} \wedge \texttt{source(y,i)} \wedge \texttt{destination(z,i)})$$

When instantiated, this yields $O(3n^2)$ propositions rather than $O(n^4)$ propositions. This technique can also be viewed as a kind of "lifting." The blocks world problems described below use these kind of linear encodings.

State-Based Encodings

The ability to express partially-ordered plans by a single model gives Graphplan a powerful performance advantage. On the other hand, we have seen that the STRIPS-style input notation has many expressive limitations. We have developed a methodology that we call "general state-based encodings", which enjoys the advantages of the two previous approaches, as well as incorporating further representational refinements.

We use the term "state-based" because it emphasizes the use of axioms that assert what it means for each individual state to be valid, and gives a secondary role to the axioms describing operators. For example, in the blocks world, the state axioms assert that only one block can be on another, every block is on something, a block cannot both be clear and have something on it, *etc.* In the logistics domain, state axioms include assertions that each transportable object can only be in a single truck, and that a truck is only at a single location.

Given that the state axioms force each state to be internally consistent, it turns out that only a relatively small number of axioms are needed to describe state transitions, where each transition can be the result of the application of any number of mutually non-conflicting actions. These axioms describe what it means for a fact to change its truth value between states.

One way to do this is to write axioms about the possible actions that could account for each change. For example, in the logistics domain, if an instance of `in` goes from false to true, then the object must have been loaded:

$$(\neg\texttt{in(x,y,i)} \wedge \texttt{in(x,y,i+1)}) \supset \exists \texttt{z.load(x,y,z,i)}$$

This style of axiom can be seen as an instance of the "domain specific" frame axioms described by Haas (1987) and Schubert (1989). Note that classical frame axioms of the type used above for linear encodings are *not* included — in fact, they are inconsistent with parallel actions. These axioms are also similar to the "backward-chaining" axioms used in the Graphplan encodings above. The Graphplan example axiom can be rewritten as

$$(\neg\texttt{maintain(in(A,R),2)} \wedge \texttt{in(A,R,3)}) \supset (\texttt{load(A,R,L,2)} \vee \texttt{load(A,R,P,2)}).$$

This formula can be identified as an instance of the general schema, once the dummy `maintain` proposition is replaced by its precondition, `in(A,R,2)`. Finally, axioms are added that assert that actions entail both their preconditions and effects, and that conflicting actions are mutually exclusive.

As described thus far, this approach has greater expressive power than the Graphplan encodings, but is no more compact. However, the number of propositional variables in this form of encoding can be significantly decreased by using the trick of reducing the arity of predicates, as described in the previous section. Furthermore, many of the axioms relating actions to their preconditions and effects can be safely eliminated, because the strong state consistency axioms propagate the consequences of the remaining assertions. This process of eliminating propositions and simplifying axioms can be carried to the extreme of *completely eliminating propositions that refer to actions*! Only fluents are used, and the axioms directly relate fluents betweens adjacent state. We have done this for the logistics domain, a relatively complex domain that involves moving packages between various locations using trucks and airplanes. The STRIPS-style formalization requires operators such as `load-truck`, `unload-truck`, `drive-truck`, `load-airplane`, *etc.* On the other hand, instead of using explicit `load` axioms, we use a single schema that relates the predicates `at` and `in`:

$$\begin{aligned}&\texttt{at(obj,loc,i)} \supset \\ &\quad \texttt{at(obj,loc,i+1)} \vee \\ &\quad \exists x \in \texttt{truck} \cup \texttt{airplane}. \\ &\qquad \texttt{in(obj,x,i+1)} \wedge \\ &\qquad \texttt{at(x,loc,i)} \wedge \\ &\qquad \texttt{at(x,loc,i+1)}\end{aligned}$$

In English, this simply asserts that if an object is at a location, it either remains at that location or goes into some truck or plane that is parked at that location. Another schema accounts for the state-transitions associated with unloading, by asserting that an object in a vehicle either stays in the vehicle, or becomes `at` the location where the vehicle is parked. Interestingly, no additional transition axioms at all are needed for the vehicle movement operators, `drive-truck` and `fly-airplane`, in this domain. The state validity axioms alone ensure that each vehicle is always at a single location.

A solution to a state-based encoding of a planning problem yields a sequence of states. The "missing" actions are easily derived from this sequence, because each pair of adjacent states corresponds to the (easy) problem of finding a unordered plan of length 1. (In the most general case, even finding unordered plans of length 1 is NP-hard; however, in domains we have examined so far, including the logistics and blocks world domains, there is a linear-time algorithm for finding such plans.) The initial motivation for developing this purely state-based representation was pragmatic: we wished to find very compact logical encodings, of a size that could be handled by our SAT algorithms. We achieved this goal: for example, we can use our stochastic algorithm to solve state-based encodings of logistic problems that cannot be solved by *any* other domain-independent planner of which we are aware. (For an example of high-performance planning using domain-*dependent* control heuristics for the blocks world, see Bacchus and Kabanza (1995).) Beyond these computational concerns, the encodings are interesting from a purely representational standpoint. There are no explicit frame axioms, or axioms about preconditions and effects, or axioms about conflicts between actions; everything is subsumed by simple, uniform relationships between fluents. These axiomatizations appear at least as "natural" as situation-calculus or STRIPS formalizations, and avoid many of the traditional problems those approaches encounter.

The experiments reported in this paper do not involve an automatic way of deriving state-based encodings from a STRIPS-style problem specification. The encodings we used in our experiments were created by hand, based on our understanding of the semantics of the various benchmark domains (which were, indeed, described by STRIPS operators). A separate paper (Kautz *et al.* 1996) describes our initial results on automating the process of *compiling away* the operators for a given domain. However, one could equally well take a state-based description of a domain as *primary*, and then add actions to the axioms through meaning postulates.

Experiments: Systematic versus Stochastic Search

In this section, we will discuss our experimental results. We first compare the various encoding schemes with respect to the cost of finding a plan. We then show that the solutions we obtained are optimal, by showing that no shorter plans exist.

To solve our SAT encodings, we consider both a systematic and a stochastic method. Tableau, the systematic procedure, is based on the Davis-Putnam procedure, and was developed by Crawford and Auton (1993). It's one of the fastest current complete SAT procedures (Trick and Johnson 1993; Dubois *et al.* 1996). Walksat, the stochastic procedure, is a descendant of GSAT, a randomized greedy local search method for satisfiability testing (Selman *et al.* 1992, Selman *et al.* 1994, 1996). Such stochastic local search methods have been shown to outperform the more traditional systematic methods on various classes of hard Boolean satisfiability problems. Note, however, that these procedures are inherently incomplete: they cannot *prove* that a formula is unsatisfiable.

Walksat operates as follows. It first picks a random truth assignment, and randomly selects one of the clauses in the SAT instance that is not satisfied by the assignment. It then flips the truth assignment of one of the variables in that clause, thereby satisfying the clause. However, in the process, one or more other clauses may become unsatisfied. Therefore, in deciding which variable to flip from the clause, Walksat uses a greedy bias that tends to increase the total number of satisfied clauses. Specifically, the bias picks the variable that minimizes the number of clauses that are satisfied by the current assignment, but which would become unsatisfied if the variable were flipped. Because the bias can lead the algorithm into local minima, performance is enhanced if the bias is not always applied. The best rule appears to be to always apply the bias if there is a choice that would make no other clauses become unsatisfied; otherwise, randomly apply it half the time. The procedure keeps flipping truth values until a satisfying assignment is found or until some predefined maximum number of flips is reached. In Selman *et al.* (1994, 1996), it was shown that this method significantly outperforms basic GSAT, and other local search methods such as such as simulated annealing (Kirkpatrick *et al.* 1983).

Finding Plans

Table 2 gives the computational cost of solving several hard planning problems. We consider two SAT encodings for each instance, one Graphplan-based and the other direct (linear or state-based). For our SAT encodings, we give both the timings for the systematic tableau method and for the stochastic Walksat procedure. We compare our results to those of the Graphplan system.

As mentioned in the preview of results, we considered hard instances from the rocket and the logistics domains (Blum and Furst 1995, Veloso 1992), as well as the blocks world. We noted that Graphplan has been shown to outperform Prodigy and UCPOP on the rocket problems. The logistics domain is a strictly richer environment than the rocket domain.[2] In the column marked with "time/actions", we give the length of the plan found in terms of the number of time steps. Since we allow for parallel (independent) actions, we also give the total number of actions that will lead us from the initial state to the goal. We created a state-based encoding for rocket and logistics problems, and for the blocks world used the original

[2]Preliminary data indicate that Graphplan, and thus our algorithms, outperform UCPOP on the logistics domain, as expected (Friedman 1996). However, it is important to note that UCPOP is a regression planner, and certain state-based notions are inaccessible or obscure to it. UCPOP may well prove superior on other domains, in which reasoning is more causal, and less related to topological notions.

				SAT Encoding					
	time /	Graphplan		Graphplan-Based			Direct		
problem	actions	nodes	time	vars	syst.	stoch.	vars	syst.	stoch.
rocket_ext.a	7/34	1,625	520	1,103	4.4	4.7	331	0.8	0.1
rocket_ext.b	7/30	1,701	2,337	1,179	2.8	21	351	2.5	0.2
logistics.a	11/54	2,891	6,743	1,782	6.9	29	828	—	2.7
logistics.b	13/47	3,382	2,893	2,069	6.4	47	843	—	1.6
logistics.c	13/65	4,326	—	2,809	23,061	262	1,141	—	1.9
bw_large.a	6/12	5,779	11.5	5,772	—	—	459	0.5	0.3
bw_large.b	9/18	18,069	27,115		—	—	1,087	1.5	22
bw_large.c	14/28		—		—	—	3,016	564	670
bw_large.d	18/36		—		—	—	6,764	—	937

Table 2: The computational cost of finding plans for several hard planning problems. For each instance, the optimal (minimal length) plan was found. Times in seconds. A long dash (—) indicates that the experiment was terminated after 10 hrs with no solution found, or, when we do not give the number of variables or the number of nodes, it means that the problem instance was too large to fit into main memory. The rocket and logistic direct encodings are state-based, and the bw (blocks world) direct encodings are linear.

linear encodings from Kautz and Selman (1992). Before applying the solvers, all of the SAT instances were first simplified by a linear-time algorithm for unit propagation, subsumption, and deletion of unit clauses. Table 2 gives the number of variables in each instance *after* simplification.

The results for "rocket_ext.a" show the general trend. The direct encodings are the most compact, and can be solved many times faster than the Graphplan-based SAT encodings, which is in turn are more efficient than extracting the plans directly from the planning graphs, using the Graphplan system.[3]

We also see that stochastic search (Walksat; see column marked "stoch.") often outperforms systematic search (tableau; see column marked "syst.") by an order of magnitude. Especially striking is the performance of Walksat on the state-based encodings (last column). These results strongly suggest that stochastic methods combined with efficient encoding techniques are a promising method for solving challenging classical planning problems.

As the instances become harder, the difference in performance between Walksat on the direct encodings and the other approaches becomes more dramatic. For example, see "logistic.c" and "bw_large.d." As we will discuss in the next section, all problems were solved to optimality. Thus, for the blocks world instance, bw_large.d, we found the minimal length plan of 36 operations (pickup/putdown/stack/unstack) from the initial state to the goal state. Only Walksat on the linear encoding could synthesize this plan. The problem involves 19 blocks, with 4 stacks in the initial and 3 stacks in the goal state. Note that we did not encode *any* special search control knowledge (such as, "move a block directly to a goal position, if possible"). To get a better feel for the computational difficulty of this problem, let us briefly consider some of the formal computational properties of the blocks world domain.

Optimal blocks world planning was shown to be NP-complete in 1991, but a plan within a factor two of optimal can be obtained in polynomial time (Gupta and Nau 1991, 1992, Chenoweth 1991). Gupta and Nau (1992) give an algorithm for finding such approximate solutions. The basic idea is to first move blocks to the table and then build up the goal stacks. Gupta and Nau's approximation algorithm would generate a plan with 58 operations for our instance, requiring 12 via-the-table moves. (Some blocks don't have to be moved to the table. Note that a via-the-table move generally involves an unstack, a putdown, a pickup, and a stack operation.) Selman (1994) shows that it's unlikely that we can find a better polytime approximation algorithm: All the difficulty lies in deciding how one can avoid via-the-table moves by making direct stack-to-stack moves. To do so, one has to determine which stack-to-stack moves to make and in what order. Walksat manages to eliminate 11 of the 12 via-the-table moves — leaving an optimal plan of 36 steps with only a single, unavoidable via-the-table move! We do not know of any other planning system that can optimally solve unrestricted blocks world problems of this size without using any kind of domain-specific control knowledge. Despite the fact that the blocks world domain is somewhat artificial, we are encouraged by our results because we believe that the rich interactions between operator and sub-goal sequencing, which makes the domain relatively hard, is also quite likely to be found in more practical domains, such as, for example, the softbot planning domain (Etzioni and Weld 1994). (Indeed, in the next phase of this project, we hope to apply our methods to the softbot domain.)

Finally, from the columns that give the number of nodes and number of variables, we see that direct encodings (and in particular, the state-based encodings) result in a significant reduction of the number of variables in the problem instances. Our Graphplan-based SAT encodings also have fewer variables than the number of nodes in the corresponding planning graph, because of the unit-propagation simplification described above.

Although our results are quite promising for stochastic methods, we do not mean to suggest that these methods will always outperform systematic ones. In fact, we have done some preliminary experiments on one of the artificial domains (D^1S^1) studied in Barrett and Weld (1994) and in Blum and Furst (1994). We considered the Graphplan-based encoding, and found

[3] Our timings do not include the time needed for generating the planning graph or for constructing the SAT encodings. On the harder instances, those times are just a fraction of what it takes to solve the planning graph or the SAT problems.

that the Graphplan system itself scales better than our SAT approach using either Walksat or tableau. The special structure of the domain, which is specifically designed to check the sequencing of operators, appears to steer Walksat repeatedly in the wrong direction. Tableau performs poorly because it performs a depth-first search, and the domain appears to require a breadth-first approach.[4] State-based encodings may again give better results on this domain. We also obtained some promising results on these instances using SAT encodings based on McAllester and Rosenblitt's (1991) "causal" planning formulation. In general, we expect that systematic and stochastic methods will complement each other — each having different relative strengths depending on the domain. In the next section, we'll discuss another way in which these methods complement each other.

Proving Optimality

To show that the plans we found in our previous experiments are optimal, we now show that no shorter plan exists. Table 3 gives our results. This time we can only use methods that systematically explore the space of all possible plans up to a certain size, because we have to demonstrate that shorter plans do not exist.

From the table, we see that in this case using tableau on the Graphplan-based SAT encoding is not very effective (except for "logistics.a"). Neither the Graphplan system nor tableau with direct SAT encodings strictly dominate one another; the former is superior on the logistics problems, and the latter on the blocks world problems.

None of the methods could show the inconsistency of "logistics.c" when using at most 12 time steps. Therefore, to show the optimality of a 13 step solution to "logistics.c", we constructed "logistics.b" as a strictly smaller subproblem. Graphplan was able to show that this problem does not have 12 step solution. It follows that "logistics.c" does have a 12 step solution either.

In general, our results suggests that it's harder to show the non-existence of a plan up to a certain length than it is to find such a plan if it exists. This kind of asymmetry has also been observed in several other problem domains (Selman 1995). The issue is closely related to the practical difference between solving NP and co-NP complete problems.

Tableau can show the infeasibility of a 17 time slot (34 stack/unstack) solution for "bw_large.d", while Walksat can find a 18 time slot (36 stack/unstack) plan (Table 2). No systematic approach could find the feasible solution. This demonstrates how stochastic and systematic methods are complementary: one can be used for plan synthesis and the other to determine lower-bounds on the plan length.

Conclusions

We have shown that for solving hard planning problems from several challenging domains, our approach of using linear or state-based axiomatizations and a general, stochastic satisfiability algorithm (Walksat) outperforms some of the best specialized planning algorithms by orders of magnitude. Furthermore, Walksat is often superior to good general (tableau) and specialized (Graphplan) systematic search engines on SAT encodings derived from STRIPS-style operators. These results challenge the common assumptions in AI that planning requires specialized search techniques, and that planning is an inherently systematic process. Of course, we are not ruling out the possibility that in other domains some of the specialized planning systems could prove superior. This is an important issue for further research.

We have also shown that systematic and local search algorithms complement each other well in the planning as satisfiability framework. Systematic algorithms can be used to provide a lower-bound on the length of solution plans, and then stochastic algorithms can be used to find the actual solutions. It is interesting to observe that in certain cases systematic algorithms are better at proving infeasibility than at finding solutions to problems instances of comparable size.

Finally, our experiments with different SAT encodings of planning problems indicates that much progress can be made by considering novel kinds of axiomatizations. In particular, our experience suggests that axiomatizations that concentrate on states and fluents can be more compact and easier to solve than approaches that directly encode STRIPS-style state-changing operators. Furthermore, these state-based encodings are interesting from a representational standpoint, and appear to provide clean and elegant ways to handle parallel actions and frame conditions.

[4] For a discussion of issue of depth-first version breadth-first search in variations of the Davis-Putnam procedure, see Dechter and Rish (1994).

References

Adorf, H.M., Johnston, M.D. (1990). A discrete stochastic neural network algorithm for constraint satisfaction problems. *Proc. of the Int. Joint Conf. on Neural Networks*, San Diego, CA, 1990.

Allen, J. (1991). Planning as temporal reasoning. *Proc. KR-89*, Cambridge, MA, 1991.

Bacchus, F. and Kabanza, F. (1995). Using temporal logic to control search in a forward chaining planner. *Proc. EWSP-95*, 157–169.

Backstrom, C. (1992). *Computational complexity of reasoning about plans*, Ph.D. thesis, Linkoping University, Linkoping, Sweden.

Barrett, A. and Weld, D. (1994). Partial-order planning: evaluating possible efficiency gains. *Artificial Intelligence*, 67:71-112, 1994.

Blum, A. and Furst, M.L. (1995). Fast planning through planning graph analysis. *Proc. IJCAI-95*, Montreal, Canada.

Bylander, T. (1991). Complexity results for planning. *Proc. IJCAI-91*, Sidney, Australia, 274-279.

Carbonell, J. , Blythe J., Etzioni, O., Gil, Y., Joseph, R., Kahn, D., Knoblock, C., Minton, S., Perez, A., Reilly, S., Veloso, M., Wang, X (1992). Prodigy 4.0: the manual and tutorial. CMU, CS Tech. Report CMU-CS-92-150.

Chenoweth, S.V. (1991). On the NP-hardness of the blocks world. *Proc. AAAI-91*, Anaheim, CA, 623–628.

Crawford, J.M. and Auton, L.D. (1993) Experimental Results on the Cross-Over Point in Satisfiability Problems. *Proc. AAAI-93*, Washington, DC, 21–27.

Crawford, J. and Baker, A.B. (1994). Experimental results on the application of satisfiability algorithms to

				SAT Encoding			
		Graphplan		Graphplan-Based		State-Based	
problem	time	nodes	time	vars	syst.	vars	syst.
rocket_ext.a	6	1,295	60.3	773	4.5	263	2.5
rocket_ext.b	6	1,372	68.6	849	3.5	283	7.2
logistics.a	10	2,469	1,273	1,415	80.1	728	—
logistics.b	12	3,002	2,946	1,729	—	757	—
logistics.c	12	3,826	—	2,353	—	1,027	—
bw_large.a	5/10	4,946	8.8	4,939	—	415	0.6
bw_large.b	8/16	16,590	—		—	920	1.2
bw_large.c	13/26		—		—	4,405	250
bw_large.d	17/34		—		—	5,886	25,289

Table 3: Showing the infeasibility of shorter plans. Times in seconds.

scheduling problems. *Proc. AAAI-94*, Seattle, WA.

Dechter, R. and Rish, I. (1994). Directional resolution: the Davis-Putnam procedure, revisited. *Proc. KR-94*, Bonn, Germany.

Dubois, O. , Andre, P., Boufkhad, Y., and Carlier, J. (1996). A-SAT and C-SAT. *Dimacs Series in Discrete Mathematics and Theoretical Computer Science.* (to appear)

Erol, K., Nau, D.S., and Subrahmanian, V.S. (1992). On the complexity of domain-independent planning. *Proc. AAAI-92*, 381–386.

Etzioni, O. and Weld, D. S. (1994). A softbot-based interface to the internet. *Comm. ACM*, July 1994.

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

Friedman, M. (1996). Personal communication.

Green, C. (1969). Application of Theorem Proving to Problem Solving. In *Proc. IJCAI-69*, Washington, D.C., 1969, 219–239.

Gupta and Nau (1991). Complexity results for blocks-world planning. *Proc. AAAI-91*, Anaheim, CA, 629–633.

Gupta and Nau (1992). On the complexity of blocks-world planning. *Artificial Intelligence*, **56**, 139–403.

Haas, A. (1987). The case for domain-specific frame axioms. In *The Frame Problem in Artificial Intelligence, Proceedings of the 1987 Workshop*, F.M. Brown, ed., Lawrence, KS, 1987. Morgan Kaufmann Publishers, Los Altos, CA.

Joslin, D. and Pollack, M. (1995). Passive and Active Decision Postponement in Plan Generation. In the *European Workshop on Planning (EWSP)*, Assisi, Italy, Sept. 1995.

Kautz, H. and Selman, B. (1992) Planning as Satisfiability. *Proc. ECAI-92*, Vienna, Austria, 1992, 359–363.

Kautz, H., McAllester, D., and Selman, B. (1996). Encoding Plans in Propositional Logic. In preparation.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983). Optimization by simulated annealing. *Science*, 220 (1983) 671–680.

McCarthy, J. and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, D. Michie, ed., Ellis Horwood, Chichester, England, 1969, page 463ff.

McAllester, D. and Rosenblitt, D. (1991). Systematic nonlinear planning. *Proc. AAAI-91*, Anaheim, CA.

Minton, S., Johnston, M.D., Philips, A.B., and Laird, P. (1990) Solving large-scale constraint satisfaction an scheduling problems using a heuristic repair method. *Proc. AAAI-90*, 1990, 17–24.

Minton, S., Johnston, M.D., Philips, A.B., and Laird, P. (1992) Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, (58)1–3, 1992, 161–205.

Pednault, E. (1988). Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4(4):356–372, 1988.

Penberthy, J. and Weld, D. (1992). UCPOP: A sound, complete, partial order planner for ADL. In the *Proc. KR-92*, Boston, MA, 103–114.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem proving. *Comm. ACM*, 5, 1962, 394–397.

Schubert, L. (1989). Monotonic Solution of the Frame Problem in the Situation Calculus: an Efficient Method for Worlds with Fully Specified Actions. In *Knowledge Representation and Defeasible Reasoning*, H. Kyburg, R. Loui, and G. Carlson, eds.

Selman, B. (1994). Near-Optimal Plans, Tractability, and Reactivity. *Proc. KR-94*, Bonn, Germany, 1994, 521–529.

Selman, B. (1995). Stochastic Search and Phase Transitions: AI Meets Physics. *Proc. IJCAI-95*, Montreal, Canada, 1995.

Selman, B. , Kautz, H., and Cohen, B. (1994). Noise Strategies for Local Search. *Proc. AAAI-94*, Seattle, WA, 1994, 337–343.

Selman, B., Kautz, H., and Cohen, B. (1996) Local Search Strategies for Satisfiability Testing. *Dimacs Series in Discrete Mathematics and Theoretical Computer Science.* (to appear)

Selman, B., Levesque, H., and Mitchell, D. (1992). A New Method For Solving Hard Satisfiability Problems. *Proc. AAAI-92*, San Jose, CA, 1992, 440-446.

Stone, P., Veloso, V., and Blythe, J. (1994). The need for different domain-independent heuristics. In *AIPS94*, pages 164-169, Chicago, 1994.

Trick, M. and Johnson, D. (Eds.) (1993) *Proc. DIMACS Challenge on Satisfiability Testing.* Piscataway, NJ, 1993. (DIMACS Series on Discr. Math.)

Veloso, M. (1992). Learning by analogical reasoning in general problem solving. Ph.D. Thesis, CMU, CS Techn. Report CMU-CS-92-174.