# Using Caching to Solve Larger Probabilistic Planning Problems*

**Stephen M. Majercik** and **Michael L. Littman**

Department of Computer Science
Duke University
Durham, NC 27708-0129
{majercik,mlittman}@cs.duke.edu

## Abstract

Probabilistic planning algorithms seek effective plans for large, stochastic domains. MAXPLAN is a recently developed algorithm that converts a planning problem into an E-MAJSAT problem, an $NP^{PP}$-complete problem that is essentially a probabilistic version of SAT, and draws on techniques from Boolean satisfiability and dynamic programming to solve the E-MAJSAT problem. This solution method is able to solve planning problems at state-of-the-art speeds, but it depends on the ability to store a value for each CNF subformula encountered in the solution process and is therefore quite memory intensive; searching for moderate-size plans even on simple problems can exhaust memory. This paper presents two techniques, based on caching, that overcome this problem without significant performance degradation. The first technique uses an LRU cache to store a fixed number of subformula values. The second technique uses a heuristic based on a measure of subformula difficulty to selectively save the values of only those subformulas whose values are sufficiently difficult to compute and are likely to be reused later in the solution process. We report results for both techniques on a stochastic test problem.

## INTRODUCTION

Classical artificial intelligence planning techniques can operate in large domains but, traditionally, assume a deterministic universe. Planning as practiced in operations research can operate in probabilistic domains, but classical algorithms for solving Markov decision processes (MDPs) and partially observable MDPs are capable of solving problems only in relatively small domains. Research in probabilistic planning aims to explore a middle ground between these two well-studied extremes with the hope of developing systems that can reason efficiently about plans in large, uncertain domains.

In this paper we describe MAXPLAN, a new approach to probabilistic planning. MAXPLAN converts a planning problem into an E-MAJSAT problem, an $NP^{PP}$-complete problem that is essentially a probabilistic version of SAT, and draws on techniques from Boolean satisfiability and dynamic programming to solve the resulting E-MAJSAT problem. In the first section, we discuss complexity results that motivated our research strategy, and compare MAXPLAN to SATPLAN, a similar planning technique for deterministic domains. The next three sections summarize our earlier work on MAXPLAN and describe the details of its operation: the planning domain representation, the conversion of problems to E-MAJSAT form and the algorithm for solving these E-MAJSAT problems, and some comparative results. The next section introduces a framework for using caching in our solver and shows how good caching strategies can greatly increase the size of problems the solver can handle without substantially altering the scaling properties of the time it takes to solve them. The final sections discuss future work and conclusions.

## COMPLEXITY RESULTS

A probabilistic planning domain is specified by a set of states, a set of actions, an initial state, and a set of goal states. The output of a planning algorithm is a controller for the planning domain whose objective is to reach a goal state with sufficiently high probability. In its most general form, a plan is a *program* that takes as input observable aspects of the environment and produces actions as output. We classify plans by their *size* (the number of internal states) and *horizon* (the number of actions produced *en route* to a goal state). In a *propositional* planning domain, states are specified as assignments to a set of propositional variables.

If we place reasonable bounds—polynomial in the size of the planning problem—on both plan size and plan horizon, the planning problem is $NP^{PP}$-complete (Littman, Goldsmith, & Mundhenk 1998) (perhaps easier than PSPACE-complete) and may be amenable to heuristics. Littman, Goldsmith, & Mundhenk (1998) provide a survey of relevant results. Membership in this complexity class suggests a solution strategy analogous to that of SATPLAN (Kautz & Selman 1996), a successful deterministic planner that converts a planning problem into a satisfiability (SAT) problem and solves the SAT problem instead. In the same way that deterministic planning can be expressed as the NP-complete problem SAT, probabilistic planning can be expressed as the $NP^{PP}$-complete problem E-MAJSAT (Littman, Goldsmith, & Mundhenk 1998):

Given a Boolean formula with *choice variables* (variables whose truth status can be arbitrarily set)

* Appears in the *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 954—959, AAAI Press/MIT Press, 1998.

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.

As we will discuss below, the choice variables can be made to correspond to a possible plan, while the chance variables can be made to correspond to the uncertainty in the planning domain. Thus, our research strategy is to show that we can efficiently turn planning problems into E-MAJSAT problems, and then focus our efforts on finding algorithms to solve the E-MAJSAT problems.

## PROBLEM REPRESENTATION

A planning domain $M = \langle \mathcal{S}, s_0, \mathcal{A}, \mathcal{G} \rangle$ is characterized by a finite set of states $\mathcal{S}$, an initial state $s_0 \in \mathcal{S}$, a finite set of actions $\mathcal{A}$, and a set of goal states $\mathcal{G} \subseteq \mathcal{S}$. Executing action $a$ in a state $s$ results in a probabilistic transition to a new state. The objective is to choose a sequence of actions to move from the initial state $s_0$ to one of the goal states with probability above some threshold $\theta$. In this work, we assume a completely unobservable domain—the effects of previous actions cannot be used in selecting the current action; thus, optimal plans are sequences of actions.

MAXPLAN represents planning domains in the sequential-effects-tree (ST) representation (Littman 1997). For each action, there is an ordered set of decision trees, one for each proposition, describing how the propositions change as a function of the state and action, perhaps probabilistically. A formal description of ST is available (Littman 1997); a brief example follows.

SAND-CASTLE-67 is a simple probabilistic planning domain concerned with building a sand castle at the beach (Figure 1). The domain has four states, described by combinations of two Boolean propositions, **moat** and **castle** (propositions appear in boldface). The proposition **moat** signifies that a moat has been dug; **castle** signifies that the castle has been built. In the initial state, both **moat** and **castle** are `False`, and the goal set is {**castle**} (a goal state is any state in which all the propositions in the goal set are `True`).

There are two actions: dig-moat and erect-castle (actions appear in sans serif). For brevity, we describe only the second decision tree (effect on **moat**) of the erect-castle action. There is no effect when **moat** is `False` (right branch) since erect-castle cannot dig a moat. But trying to erect a castle may destroy an existing moat (left branch). If the castle existed when erect-castle was selected, the moat remains intact with probability 0.75. If the castle did not exist, but erect-castle creates a new castle (**castle:new** refers to the value of **castle** *after* the first decision tree is evaluated), **moat** remains `True`. If erect-castle fails to build a castle, **moat** remains `True` with probability 0.5.

## CONVERSION AND SOLUTION

MAXPLAN's conversion unit is a LISP program that takes a planning problem in ST form and a plan horizon $N$ and produces an E-MAJSAT CNF formula. This formula has the property that, given an assignment to the choice variables (an $N$-step plan), the probability of a satisfying assignment with respect to the chance variables is the probability of success for that plan. The converter operates by time indexing each proposition and action, and making satisfaction equivalent to the enforcement of the following conditions:

- the initial conditions hold at time 0 and the goal conditions at time $N$,
- actions at time $t$ are mutually exclusive ($1 \leq t \leq N$),
- proposition $p$ is `True` at time $t$ if it was `True` at time $t-1$ and the action taken at $t$ does not make it `False`, *or* the action at $t$ makes $p$ `True` ($1 \leq t \leq N$).

The first two conditions are not probabilistic and can be encoded in a straightforward manner (Kautz & Selman 1996), but the third condition is complicated by the fact that chance variables sometimes intervene between actions and their effects on propositions.

The conversion process is detailed elsewhere (Majercik & Littman 1998); here we summarize some key facts. The total number of variables in the formula is $V = (A+P+R)N+P$, where $A$, $P$, and $R$ are the number of actions, propositions, and random propositions, respectively. The total number of clauses is bounded by a low-order polynomial in the size of the problem: $2P+\left(\binom{A}{2}+1\right)N+2N\sum_{i=1}^{A} L_i$, where $L_i$ is the number of leaves in the decision trees of action $i$. The average clause size is dominated by the average path length of all the decision trees.

Note that fixing a plan horizon does not prevent MAXPLAN from solving planning problems where the horizon is unknown. By using *iterative lengthening*, a process in which successive instances of the planning problem with increasing horizons are solved, the optimal plan horizon can be discovered dynamically. This, in fact, was the process used in the MAXPLAN/BURIDAN comparison described later. We have not yet determined the feasibility of *incremental iterative lengthening*, a more

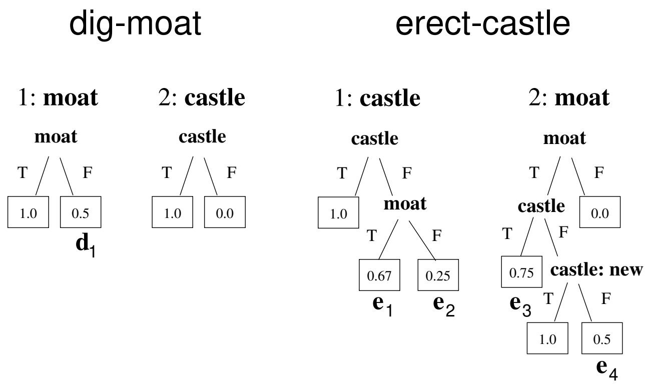


Figure 1: Sequential-Effects-Tree Representation For SAND-CASTLE-67

sophisticated approach, in which the current instance of the planning problem with horizon $N$ is incrementally extended to the instance with horizon $N+1$ and earlier results are reused to help solve the extended problem.

### Solver

An algorithm for solving the E-MAJSAT problem produced by the conversion described above needs to find the assignment to the choice variables that maximizes the probability of a satisfying assignment; for plan-generated formulas, such an assignment is directly interpretable as an optimal straight-line plan. Our algorithm is based on an extension of the Davis-Putnam-Logemann-Loveland (DPLL) procedure for determining satisfiability. Essentially, we use DPLL to determine all possible satisfying assignments, sum the probabilities of the satisfying assignments for each possible choice-variable assignment, and then return the choice-variable assignment (plan) with the highest probability of producing a satisfying assignment (goal satisfaction).

Determining all the satisfying assignments can be envisioned as constructing a binary tree in which each node represents a choice (chance) variable, and the two subtrees represent the two possible remaining subformulas given the two possible assignments (outcomes) to the parent choice (chance) variable. At any leaf of this tree, the chain of variable bindings leading to the root is sufficient to evaluate the Boolean formula—at least one literal in each clause is `True` or all literals in one clause are `False`. It is critical to construct an efficient tree to avoid evaluating an exponential number of assignments. In the process of constructing this tree:

- an *active variable* is one that has not yet been assigned a truth value,
- an *active clause* is one that has not yet been satisfied by assigned variables,
- the *current CNF subformula* is uniquely specified by the current sets of active clauses and variables, and
- the *value* of a CNF subformula is

$$\max_{\mathbf{D}} \sum_{\mathbf{S}} \left( \prod_{v_i \in \mathbf{CH}} \pi_i^{tr(v_i)} (1-\pi_i)^{1-tr(v_i)} \right),$$

  where $\mathbf{D}$ is the set of all possible assignments to the active choice variables, $\mathbf{S}$ is the set of all satisfying assignments to the active chance variables, $\mathbf{CH}$ is the set of all active chance variables, $tr(v_i) \in \{0,1\}$ is the truth value of $v_i$ ($0 =$ `False`, $1 =$ `True`), and $\pi_i$ is the probability that $v_i$ is `True`.

In general, a brute-force computation of a CNF subformula value is not feasible. To prune the number of assignments that must be considered, full DPLL uses several variable selection heuristics (Majercik & Littman 1998). Our modified DPLL uses only one of these; we select, whenever possible, a variable that appears alone in an active clause and assign the appropriate value (unit propagation, **UNIT**). For chance variable $i$, this decreases the success probability by a factor of $\pi_i$ or $1-\pi_i$. This can be shown to leave the value of the current CNF formula unchanged.

When there are no more unit clauses, we must split on an active variable (always preferring choice variables). If we split on a choice (chance) variable, we return the maximum (probability weighted average) of assigning `True` to the variable and recurring or assigning `False` and recurring. The splitting heuristic used is critical to the algorithm's efficiency; experiments (Majercik & Littman 1998) indicate that splitting on the variable that would appear earliest in the plan—time-ordered splitting (**TIME**)—is a very successful heuristic.

The solver still scaled exponentially with plan horizon, however, even on some simple plan-evaluation computations. The fact that a simple dynamic-programming plan evaluation algorithm can be created that scales linearly with plan horizon (see the following section) led us to incorporate dynamic programming into the solver in the form of memoization: the algorithm stores the values of solved subformulas for possible reuse. This greatly extends the size of the plan we can feasibly evaluate (Majercik & Littman 1998).

We tested modified DPLL (**UNIT/TIME**) on the full plan-generation problem in SAND-CASTLE-67 for plan horizons ranging from 1 to 10. Optimal plans found by MAXPLAN exhibit a rich structure: beyond a horizon of 3, the horizon $i$ plan is not a subplan of the horizon $i+1$ plan. The optimal 10-step plan is D-E-D-E-E-D-E-D-E-E (D = dig-moat, E = erect-castle). This plan succeeds with near certainty (probability 0.9669) and MAXPLAN finds it in approximately 8 seconds on a Sun Ultra-1 Model 140 with 128 Mbytes of memory.

## COMPARISONS

We compared MAXPLAN to three other planning techniques (Majercik & Littman 1998):

- BURIDAN (Kushmerick, Hanks, & Weld 1995), a classical AI planning technique that extends partial-order planning to probabilistic domains,
- Plan enumeration with dynamic programming for plan evaluation (ENUM), and
- Dynamic programming (incremental pruning) to solve the corresponding finite-horizon partially observable MDP (POMDP).

There are other important comparisons that should be made; belief networks and influence diagrams (Pearl 1988) are closely related to our work and there are efficient techniques for evaluating them, but their performance relative to MAXPLAN is unknown.

Table 1 summarizes the performance of MAXPLAN on two probabilistic planning problems described by Kushmerick, Hanks, & Weld (1995). Also shown are the running times reported for the two versions of BURIDAN that performed best on these two problems. MAXPLAN does as well as BURIDAN on SLIPPERY GRIPPER and has an order of magnitude advantage on BOMB/TOILET.

| | Planning time in CPU secs | | |
|---|---|---|---|
| | MAXPLAN | BURIDAN | |
| | Modified DPLL | Forward | Forward-Max |
| SLIPPERY GRIPPER | 0.4 | 4.5 | 0.5 |
| BOMB/TOILET | 0.3 | 6.9 | 7.0 |

Table 1: Comparison of running times for two probabilistic planning domains

We also compared the scaling behavior of MAXPLAN to that of ENUM and POMDP on two problems: SAND-CASTLE-67 and DISARMING-MULTIPLE-BOMBS (Majercik & Littman 1998). MAXPLAN scales exponentially ($O(2.24^N)$) as the horizon increases in SAND-CASTLE-67. ENUM, as expected, also scales exponentially ($O(2.05^N)$). But POMDP, remarkably, scales linearly as the horizon increases. We see much different behavior in DISARMING-MULTIPLE-BOMBS, a problem that allows us to enlarge the state space without increasing the optimal plan horizon. As the state space increases, both ENUM and POMDP scale exponentially ($O(7.50^N)$ and $O(3.50^N)$ respectively), while MAXPLAN's solution time remains constant (less than 0.1 second) over the same range.

## CACHING IN THE SOLVER

Further tests of MAXPLAN, however, uncovered a significant problem. Because it stores the value of all subformulas encountered in the solution process, the algorithm is very memory intensive and, in fact, is unable to find the best plan with horizon greater than 15 due to insufficient memory. The situation is illustrated in Figure 2, which compares the performance of full DPLL without memoization to that of modified DPLL (**UNIT/TIME**) with memoization. Note that this is a log plot and that we show the results starting with a plan horizon of 4 since the asymptotic behavior of the algorithm does not become clear until this point.

The top plot shows the performance of full DPLL without memoization. We can extrapolate to estimate performance on larger problems (dotted line), but solution times become prohibitively long. The lower plot ending with an "X" shows the performance of modified DPLL with memoization. The much lower slope of this line (2.24 compared to 3.82 for full DPLL without memoization) indicates the superior performance of this algorithm, but the "X" indicates that no extrapolation is possible beyond horizon 15 due to insufficient memory. In fact, performance data for the horizon 15 plan already indicate memory problems. The wall-clock time is more than three times the CPU time, indicating that the computation is I/O-bound. Memoization allows the algorithm to run orders of magnitude faster but ultimately limits the size of problems that can be solved.

### LRU Caching

Our solution is to treat the fixed amount of memory available as a *cache* for subformulas and their values.

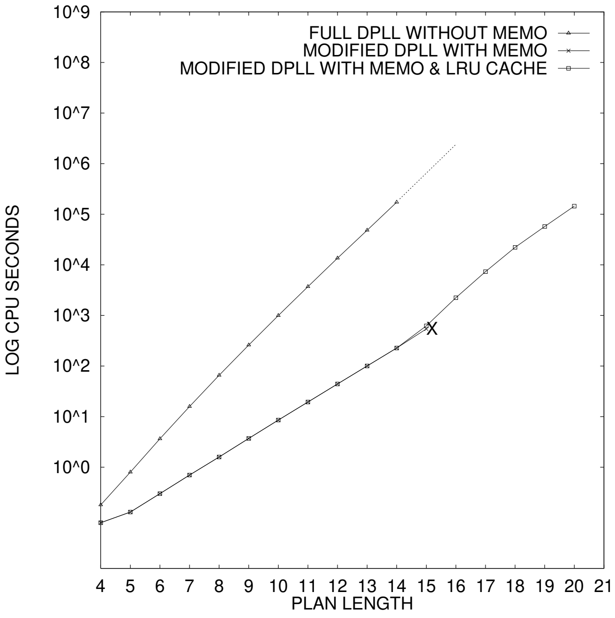


Figure 2: Full DPLL without memoization v. modified DPLL with memoization v. modified DPLL with memoization and LRU caching for SAND-CASTLE-67

Given a cache size appropriate for the amount of memory on the machine running the algorithm, the problem becomes one of finding the best replacement policy for this cache. We compared two well-known cache replacement policies: first-in-first-out (FIFO) and least-recently-used (LRU). Both were implemented through a linked list of subformulas maintained in the order they were saved. When the cache is full we merely remove the subformula at the head of the list. Under an LRU policy, however, whenever we use a subformula we move it to the end of the linked list. As shown in Figure 3, the LRU cache outperforms the FIFO cache by an average of approximately 18% across the entire range of cache sizes for the 10-step SAND-CASTLE-67 plan.

We tested the LRU caching technique for generating SAND-CASTLE-67 plans with horizons ranging from 1 to 20. For plan horizons from 1 to 15, the cache was made as large as possible without producing significant I/O problems. For larger problems—with larger subformulas to be saved—we calculated cache size so as to keep total cache bytes approximately constant. These results are shown in Figure 2; the performance plot for LRU caching is essentially coincident with the modified

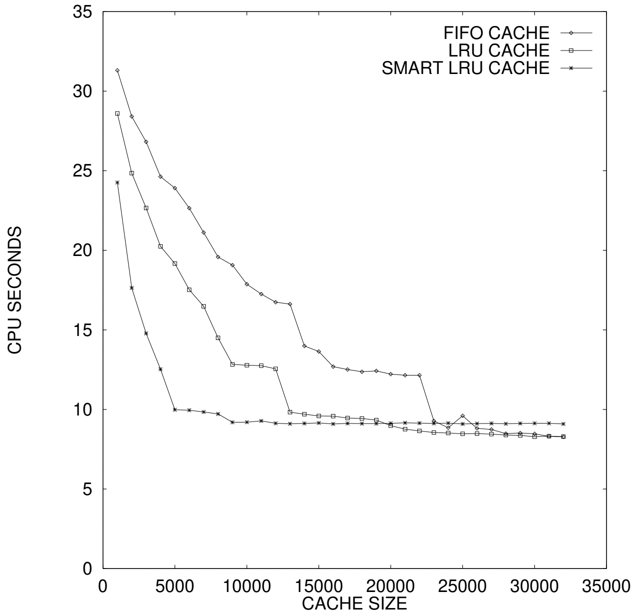


Figure 3: FIFO v. LRU v. Smart LRU for the 10-step plan for SAND-CASTLE-67

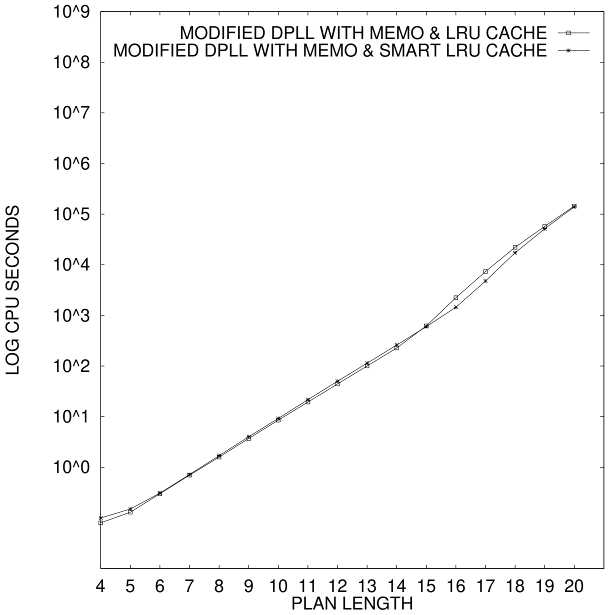


Figure 4: Modified DPLL with memoization and LRU caching v. modified DPLL with memoization and smart LRU caching for SAND-CASTLE-67

DPLL plot up to a plan horizon of 15. For longer plan horizons, LRU caching allows us to break through the memory insufficiency that blocked us before (the "X") yet retain a significant degree of the improved performance that memoization gave us. Solution times for full DPLL without memoization grow as $O(3.82^N)$, where $N$ is the plan horizon. In contrast, modified DPLL with memoization scales as only $O(2.24^N)$, but can only solve problems of a bounded size. Modified DPLL with memoization and caching behaves like modified DPLL with memoization for small $N$, and then appears to grow as $O(2.96^N)$ once $N$ is large enough for the cache replacement policy to kick in. Thus, the rate of increase for the variant with caching exhibits a growth rate comparable to that of modified DPLL with memoization while eliminating that algorithm's limitation on problem size—it trades away some performance for the ability to solve larger problems. As reported above, ENUM scales as $O(2.05^N)$ without any memory problems.

### Smart LRU Caching

We can sometimes improve performance by being selective about the subformulas we cache. The hierarchical relationship among subformulas makes it redundant to save every subformula, but which ones do we save? Large subformulas are unlikely to be reused, and small subformulas, whose value we could quickly recompute, yield little time advantage. This suggests a strategy of saving mid-size subformulas, but experiments indicate that this approach fails for the SAND-CASTLE-67 problem. Another approach is to save subformulas based on difficulty, defined as the number of recursive function calls required to solve that subformula. We experimentally determined that the optimal difficulty range for the SAND-CASTLE-67 problem was 5 to 14. Saving only those subformulas with a difficulty in this range allowed us to find the best 10-step plan for the SAND-CASTLE-67 problem with only an approximate 10% increase in CPU time, and this performance was nearly constant over a broad range of cache sizes (10,000 up to the maximum usable cache of 32,000); see Figure 3.

Applying this "smart" LRU caching technique to plan generation in the SAND-CASTLE-67 domain, we obtained as much as a 37% decrease in CPU seconds compared to straight LRU caching (Figure 4). Unfortunately, the advantages of smart caching seem to disappear as plan horizon increases. Also, it is not obvious how this technique could be exploited in a practical algorithm since we performed extensive tests to determine the optimal difficulty range—tests with other domains indicate that this range is problem dependent.

## FUTURE WORK

In addition to the problems described above, we have tested MAXPLAN on several variants of BOMB/TOILET (with clogging and asymmetric actions). But the scaling behavior of MAXPLAN with respect to the size of the state space and the number of actions in the planning domain is largely unexplored. DISARMING-MULTIPLE-BOMBS shows that MAXPLAN can find *simple* solutions efficiently as the state space grows, but we need to test our planner on more complicated domains.

Although our improvements have increased the efficiency of MAXPLAN by orders of magnitude over our initial implementation, more work needs to be done before MAXPLAN can be successfully applied to large-scale probabilistic planning problems. Efficiency could probably be improved by using a better CNF encoding of the planning problem and by using more sophisticated data structures for storing the CNF formulas.

Another promising area is splitting heuristics. Our time-ordered splitting heuristic does not specify an ordering for variables associated with the same time step. A heuristic that addresses this issue could provide a significant performance gain in real-world problems that have a large number of variables at each time step.

When we convert a planning problem to an E-MAJSAT problem, the structure of the planning problem becomes obscured, making it difficult to use our intuition to develop search control heuristics or to prune plans. Although the current E-MAJSAT solver uses heuristics to prune the number of *assignments* that must be considered, this does not directly translate into pruned *plans*. We would like to be able to use our knowledge and intuition about the planning process to develop search control heuristics and plan-pruning criteria. Kautz & Selman (1998), for example, report impressive performance gains resulting from the incorporation of domain-specific heuristic axioms in the CNF formula for deterministic planning problems.

Our experiments with caching yielded promising results, but there are three areas where further work needs to be done. First, although smart LRU caching yielded significant performance improvements, it is not practical unless we can determine the optimal difficulty range for subformulas to be cached without extensive testing. Second, it appears that we could do even better than smart caching. Performance on the 10-step SAND-CASTLE-67 problem deteriorates significantly if the cache size is less than approximately 5000 (Figure 3), yet we know that there are only 1786 distinct subformulas that are reused in the solution process. A better heuristic for selecting subformulas to save could improve the benefits of this technique. Finally, a more sophisticated cache replacement policy could yield additional performance gains.

Approximation techniques need to be explored. Perhaps we can solve an approximate version of the problem quickly and then explore plan improvements in the remaining available time, sacrificing optimality for "anytime" planning and performance bounds. This does not improve worst-case complexity, but is likely to help for typical problems.

Finally, the current planner assumes complete unobservability and produces an optimal straight-line plan; a practical planner must be able to represent and reason about *conditional plans*, which can take advantage of circumstances as they evolve, and *looping plans*, which can express repeated actions compactly (Littman, Goldsmith, & Mundhenk 1998).

## CONCLUSIONS

We have described an approach to probabilistic planning that converts the planning problem to an equivalent E-MAJSAT problem—a type of Boolean satisfiability problem—and then solves that problem. The solution method we have devised uses an adaptation of a standard systematic satisfiability algorithm to find all possible satisfying assignments, along with memoization to accelerate the search process. This solution method solves some standard probabilistic planning problems at state-of-the-art speeds, but is extremely memory intensive and exhausts memory searching for moderate-size plans on simple problems.

We described two techniques, based on caching, for reducing memory requirements while still preserving performance to a significant degree. The first technique, using an LRU cache to store subformula values, is generally applicable, regardless of the specific planning problem being solved. The second technique, selectively saving subformula values in an LRU cache based on their computational difficulty, is, at present, problem dependent. But this technique demonstrates the significant performance benefits that can be obtained with smarter LRU caching.

## Acknowledgments

This work was supported in part by grant NSF-IRI-97-02576-CAREER.

## References

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. AAAI Press/The MIT Press.

Kautz, H., and Selman, B. 1998. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning*, 181–189. AAAI Press.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1-2):239–286.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The computational complexity of probabilistic plan existence and evaluation. *Journal of Artificial Intelligence Research*. To appear.

Littman, M. L. 1997. Probabilistic propositional planning: Representations and complexity. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 748–754. AAAI Press/The MIT Press.

Majercik, S. M., and Littman, M. L. 1998. MAXPLAN: A new approach to probabilistic planning. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning*, 86–93. AAAI Press.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. San Mateo, CA: Morgan Kaufmann.