Approximate Planning in the Probabilistic-Planning-as-Stochastic-Satisfiability Paradigm[*]

Stephen M. Majercik and **Michael L. Littman**
Department of Computer Science
Duke University
Durham, NC 27708-0129
{majercik,mlittman}@cs.duke.edu

Abstract

ZANDER is a state-of-the-art probabilistic planner that extends the probabilistic-planning-as-stochastic-satisfiability paradigm to support contingent planning in domains where there is uncertainty in the effects of the agent's actions and where the scope and accuracy of the agent's observations may be insufficient to establish the agent's current state with certainty (Majercik & Littman 1999). We describe ZANDER and then discuss an approximation technique we are developing that will help us to scale up our SSAT-based technique to large planning problems. We report results using this approximation algorithm on random SSAT problems and discuss issues that arise in the application of this algorithm to SSAT encodings of planning problems.

Introduction

Planning is a critical activity in space exploration efforts. Uncertainty in the various domains—especially partial observability—makes it impossible to rely on a simple straight-line plan. One approach to dealing with this uncertainty is to construct a plan as if the environment were deterministic, and then replan quickly if the plan fails. If the uncertainty in the environment is quantifiable, however, we can use that knowledge to construct a plan with contingencies so that there is less likelihood of having to replan.

Our work focuses on the latter approach. ZANDER is a probabilistic planner that extends the probabilistic-planning-as-stochastic-satisfiability paradigm to support contingent planning in domains where there is uncertainty in the effects of the agent's actions and where the scope and accuracy of the agent's observations may be insufficient to establish the agent's current state with certainty (Majercik & Littman 1999). ZANDER solves a planning problem by converting it into an instance of stochastic satisfiability (SSAT), a type of satisfiability problem in which some of the variables have probabilities attached to them. A solution to the SSAT problem yields a plan that has the highest probability of succeeding. Initial results have been very encouraging—ZANDER operates at state-of-the-art speeds on planning problems drawn from the literature—and we are now developing techniques based on this paradigm that will allow us to scale up to large, real-world problems. A natural approach is an approximation technique and, in this paper, we describe efforts to develop such an algorithm for solving the SSAT encodings of planning problems generated by our planner.

We envision two possible applications of our planning technique (Brooks 2000). First, our planner has the potential to accept a list of requests for use of a particular spacecraft, organize these requests into a viable sequence of activities, and convert these requests into input for other elements of the uplink data system. Second, our planner could be used as part of an on-board fault protection system in a spacecraft. Given a fault condition, the planner would observe the condition of the spacecraft, plan more diagnostic tests if necessary, create a contingent plan to correct the fault and, finally, replan any interrupted or unexecuted activities and continue the sequence.

In the remainder of this section, we describe our domain representation, the SSAT framework, and how contingent planning problems can be encoded as SSAT problems. After a brief description of ZANDER, we describe our approximation algorithm, report results on a large number of random SSAT problems, and discuss issues that arise in applying this algorithm to SSAT encodings of planning problems. We conclude with an assessment of the strengths and weaknesses of this approach.

[*] Appears in the *Proceedings of the Second International Workshop on Planning and Scheduling for Space*, pages 60-66, 2000

Probabilistic Planning Representation

zander uses a propositional representation for planning problems. A planning problem is described using a finite set P of n distinct *propositions*, each of which may be `True` or `False` at any (discrete) time t. A *state* is an assignment of truth values to P. A probabilistic initial state is specified by a set of decision trees, one for each proposition. *Goal states* are specified by a partial assignment G to the set of propositions; any state that extends G is considered to be a goal state. Each of a finite set A of *actions* probabilistically transforms a state at time t into a state at time $t + 1$ and so induces a probability distribution over the set of all states. In this work, the effect of each action on each proposition is represented as a separate decision tree (Boutilier & Poole 1996). For a given action a, each of the decision trees for the different propositions are ordered, so the decision tree for one proposition can refer to both the new and old values of previous propositions. The leaves of a decision tree describe how the associated proposition changes as a function of the state and action. A subset of the set of propositions is the set of *observable propositions*, each of which has, as its basis, a proposition that represents the actual status of the thing being observed. The planning task is to find a plan that selects an action for each step t as a function of the value of observable propositions for steps before t. We want to find a plan that maximizes (or exceeds a user-specified threshold for) the probability of reaching a goal state.

Stochastic Satisfiability

In the deterministic satisfiability problem, or SAT, we are given a Boolean formula and wish to determine whether there is some assignment to the variables in the formula that results in the formula evaluating to `True`. Papadimitriou (1985) explored an extension of SAT in which a random quantifier is introduced. The stochastic SAT (SSAT) problem is to evaluate a Boolean formula in which existential and random quantifiers alternate:

$$\exists x_1, \rotatebox[origin=c]{180}{R} x_2, \exists x_3, \ldots, \exists x_{n-1}, \rotatebox[origin=c]{180}{R} x_n (E[\phi(\mathbf{x})] \geq \theta).$$

In words, this formula asks whether there is a value for x_1 such that, *for random* values of x_2 (choose 0 or 1 with equal probability), there exists a value of $x_3 \ldots$ such that the *expected* value, or probability of satisfaction, of the Boolean formula $\phi(\mathbf{x})$ is at least a threshold $0 \leq \theta \leq 1$. In our SSAT problems, we will allow *blocks* of existential and random quantifiers to alternate. Furthermore, we will allow *annotated random quantifiers* such as $\rotatebox[origin=c]{180}{R}^{0.2}$, which takes on value `True` with probability 0.2 and `False` with probability 0.8. The specification of an SSAT problem consists of the Boolean formula $\phi(\mathbf{x})$, the probability threshold θ, and the ordering of the quantifiers. In what follows, we will refer to existentially quantified variables in the SSAT formula as *existential variables* and randomly quantified variables as *randomized variables*.

Encoding Planning Problems

In an SSAT formula, the value of an existential variable x can be selected on the basis of the values of all the variables to x's left in the quantifier sequence. This suggests a way of mapping contingent planning problems to stochastic satisfiability: encode the contingent plan in the variable ordering associated with the SSAT formula. By alternating blocks of existential variables that encode actions and blocks of randomized variables that encode observations, we can condition the value chosen for any action variable on the possible values for all the observation variables that appear earlier in the ordering. A generic SSAT encoding for contingent plans appears in Figure 1.

The quantifiers fall into three segments: a plan-execution history, the domain uncertainty, and the result of the plan-execution history given the domain uncertainty. The plan-execution-history segment is an alternating sequence of existential-variable blocks (one for each action choice) and randomized-variable blocks (one for each set of possible observations at a time step).

The domain uncertainty segment is a single block containing all the randomized variables that modulate the impact of the actions on the observation and state variables. These variables are associated with random quantifiers; when we consider a variable that represents uncertainty in the environment, we want to take the probability weighted average of the success probabilities associated with the two possible settings of the variable.

The result segment is a single block containing all the non-observation state variables. These variables are associated with existential quantifiers, indicating that we can choose the best truth setting for each variable. In reality, all such "choices" are forced by the settings of the action variables in the first segment and the chance variables in the second segment. If these forced choices are compatible, then the preceding plan-execution history is possible and has a non-zero probability of achieving the goals. Otherwise, either the plan-execution history is impossible, given the effects of the actions, or it has a zero probability of achieving the goals.

An attractive feature of this planning technique is that it is straightforward to add additional constraints to the SSAT encoding of the planning prob-

$$\overbrace{\exists x_{1,1},\ldots,\exists x_{1,c_1}}^{\text{first action}}\overbrace{\text{ꓤ} w_{1,1},\ldots,\text{ꓤ} w_{1,c_2}}^{\text{first observation}}\cdots\overbrace{\text{ꓤ} w_{n-1,1},\ldots,\text{ꓤ} w_{n-1,c_2}}^{\text{last observation}}\overbrace{\exists x_{n,1},\ldots,\exists x_{n,c_1}}^{\text{last action}}$$

$$\overbrace{\text{ꓤ}^{\rho_1} z_1,\ldots,\text{ꓤ}^{\rho_{c_4}} z_{c_4}}^{\text{random outcomes}}\overbrace{\exists y_1,\ldots,\exists y_{c_3}}^{\text{the state}}(E[\phi(\mathbf{x})]\geq\theta).$$

c_1 = number of variables it takes to specify a single action (the number of actions),
c_2 = number of variables it takes to specify a single observation,
c_3 = number of state variables (one for each proposition at each time step), and
c_4 = number of chance variables (one for each possible stochastic outcome at each time step).

Figure 1: A contingent planning problem can be encoded as an instance of SSat.

lem. This means, for example, that human judgments about activities that *must* be performed can easily be enforced.

ZANDER

We briefly describe zander, our SSat-based probabilistic planner. Details are available elsewhere (Majercik & Littman 1999). zander must find an assignment *tree* that specifies the optimal existential-variable assignment given all possible settings of the observation variables. The most basic variant of the solver follows the variable ordering exactly, constructing a binary tree of all possible assignments. Each node in the tree contains a variable under consideration, and each path through the tree describes a plan-execution history, an instantiation of the domain uncertainty, and a possible setting of the state variables. An observation variable is a branch point; the optimal assignment to the remaining variables will, in general, be different for different values of this variable.

The solver does a depth-first search of the tree, constructing a solution subtree by calculating, for each node, the probability of a satisfying assignment given the partial assignment so far. For an existential variable, this is a maximum probability and produces no branch in the solution subtree; the solver notes which value of the variable yields this maximum. For a randomized variable, the probability will be the probability weighted average of the success probabilities for that node's subtrees and will produce a branch point in the solution subtree. The solver finds the optimal plan by determining the subtree with the highest success probability.

We use three pruning techniques to avoid checking every possible truth assignment. Whenever an existential or randomized variable appears alone in an active clause, *unit propagation* assigns the forced value to that variable. Whenever an existential variable appears always negated or always not negated in all active clauses, *variable purification* assigns the appropriate value to that variable. *Thresholding* allows us to prune plans based on a prespecified threshold probability of success (i.e. find a plan whose probability of success meets or exceeds the threshold probability).

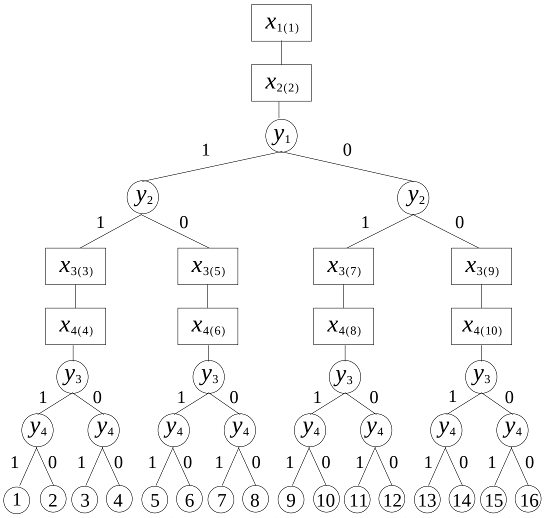
An SSAT Approximation Algorithm

zander performs at state-of-the-art speeds on problems drawn from the literature (Majercik & Littman 1999). This is encouraging since there are a number of potential improvements to zander that have shown promise for scaling up to larger problems (better data structures to optimize the application of heuristics, more compact and efficient SSat encodings, encoding domain knowledge, memoization for contingent planning, using learning to accelerate the solution process, and more sophisticated splitting heuristics). In order to scale up to even larger problems, however, it may be necessary to develop approximation techniques. In this section we describe one possible approach.

Our approximation algorithm randevalssat takes an SSat formula and returns an approximation of its value, along with the policy that approximately produces this value. The algorithm uses a *policy tree* representation. Policy trees include multiple copies of each existential variable—one for each possible assignment to the randomized variables that precede it in the quantifier ordering—thus emphasizing the fact that the value of each existential variable can be a function of the values of the preceding randomized variables. Figure 2 shows the policy tree for the following SSat instance:

$$\exists x_1,\exists x_2,\text{ꓤ} y_1,\text{ꓤ} y_2,\exists x_3,\exists x_4,\text{ꓤ} y_3,\text{ꓤ} y_4,$$
$$E[(\overline{x_1}\vee x_3\vee\overline{y_3})(\overline{x_2}\vee\overline{x_4}\vee y_2)(x_3\vee y_1\vee\overline{y_4})]\geq\theta.$$

Existential variables, and their copies, are called *decision variables* and are shown as rectangular *decision nodes* in the policy tree. To indicate that the value of each instance of a copied variable can be set independently, these variables are renumbered in the policy tree (parenthesized subscripts).

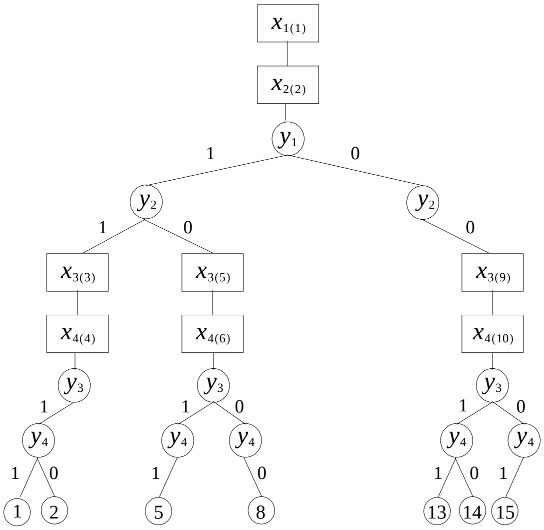


Original Formula: $(\bar{x}_1 \vee x_3 \vee \bar{y}_3)(\bar{x}_2 \vee \bar{x}_4 \vee y_2)(x_3 \vee y_1 \vee \bar{y}_4)$

Simplified Formulae at Leaves Given Random Variable Assignments

Leaf 1:	$(\bar{x}_1 \vee x_3)$	Leaf 9:	$(\bar{x}_1 \vee x_7)(x_7)$
Leaf 2:	$(\bar{x}_1 \vee x_3)$	Leaf 10:	$(\bar{x}_1 \vee x_7)$
Leaf 3:	Satisfied	Leaf 11:	(x_7)
Leaf 4:	Satisfied	Leaf 12:	Satisfied
Leaf 5:	$(\bar{x}_1 \vee x_5)(\bar{x}_2 \vee \bar{x}_6)$	Leaf 13:	$(\bar{x}_1 \vee x_9)(\bar{x}_2 \vee \bar{x}_{10})(x_9)$
Leaf 6:	$(\bar{x}_1 \vee x_5)(\bar{x}_2 \vee \bar{x}_6)$	Leaf 14:	$(\bar{x}_1 \vee x_9)(\bar{x}_2 \vee \bar{x}_{10})$
Leaf 7:	$(\bar{x}_2 \vee \bar{x}_6)$	Leaf 15:	$(\bar{x}_2 \vee \bar{x}_{10})(x_9)$
Leaf 8:	$(\bar{x}_2 \vee \bar{x}_6)$	Leaf 16:	$(\bar{x}_2 \vee \bar{x}_{10})$

Figure 2: A policy tree (a) represents the set of contingent choices in an SSat problem.



Original Formula: $(\bar{x}_1 \vee x_3 \vee \bar{y}_3)(\bar{x}_2 \vee \bar{x}_4 \vee y_2)(x_3 \vee y_1 \vee \bar{y}_4)$

Assignments Responsible for Leaves in Partial Decision Tree

Leaf 1:	$y_1 = 1$	$y_2 = 1$	$y_3 = 1$	$y_4 = 1$
Leaf 2:	$y_1 = 1$	$y_2 = 1$	$y_3 = 1$	$y_4 = 0$
Leaf 5:	$y_1 = 1$	$y_2 = 0$	$y_3 = 1$	$y_4 = 1$
Leaf 8:	$y_1 = 1$	$y_2 = 0$	$y_3 = 0$	$y_4 = 0$
Leaf 13:	$y_1 = 0$	$y_2 = 0$	$y_3 = 1$	$y_4 = 1$
Leaf 14:	$y_1 = 0$	$y_2 = 0$	$y_3 = 1$	$y_4 = 0$
Leaf 15:	$y_1 = 0$	$y_2 = 0$	$y_3 = 0$	$y_4 = 1$

Figure 3: Randomized local search can be applied to a partial policy tree, obtained by sampling, to provide an approximate answer for an SSat instance.

Each leaf of the policy tree represents a partial assignment consisting of an assignment to all randomized variables in the SSat formula. The *probability of a leaf* is the product of the probabilities of the outcomes along the path from the root to the leaf. A *policy* is an assignment of Boolean values to the decision variables in the policy tree. Given a policy, all the root-to-leaf paths in the policy tree represent complete assignments to the variables in the formula, each of which is either satisfying (value 1) or unsatisfying (value 0). The *value of a policy* is the weighted sum of the probabilities of the satisfied leaves. The *value of a policy tree* is the maximum over all policies of the policy values.

A systematic search for the policy with the highest value would solve the SSat problem, but this is a doubly exponential process: the size of the policy tree is exponential in the number of randomized variables and systematically searching for the best policy is exponential in the number of decision nodes in the policy tree. Instead, the algorithm randevalssat uses stochastic sampling to limit the size of the policy tree constructed and randomized local search to find the best policy for that reduced tree. This approach is similar to that of Kearns, Mansour, & Ng (1999) for choosing approximately optimal actions with high probability in infinite-horizon discounted Markov decision processes.

The algorithm constructs a *partial policy tree* by choosing a set W of w assignments to the randomized variables proportional to their probability and independently of the policy. These sampled assignments select out a set of leaves from the full policy tree, with the probability of an assignment given by its frequency of selection in the random sample. The top of Figure 3 gives a partial policy tree derived from the sample in the bottom of Figure 3 and the policy tree of Figure 2. The value of a policy is then estimated as the proportion of the w sampled leaves that are satisfied by the policy. A direct application of Chernoff bounds shows that $w = O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ samples are sufficient to be sure with probability $1-\delta$ of having an estimate no further than ϵ away from the true value (Littman, Majercik, & Pitassi 2001). Note that this number does not depend on n, m, k, or how the SSat formula or the policy was created.

As illustrated in Figure 2, the effect of a policy on a leaf can be summarized by a Boolean formula, called a *path formula*, for that leaf. Each random sample can potentially produce a different path formula, and the union of all the path formulas produces a *treeSAT* formula, which can be viewed as both a collection of clauses and a collection of path formulas. A satisfying assignment to the treeSAT formula corresponds to a setting of the decision variables that satisfies the original SSat formula along all paths in the partial policy tree. In general, this will not be possible; instead randevalssat use randomized local search to search for a treeSAT assignment that maximizes the number of path formulas that are satisfied. In our experiments, this was done by hillclimbing on an objective function that counts both clauses satisfied and path formulas satisfied. A satisfied path formula is weighted as heavily as the number of clauses in the original SSat formula, so that satisfying all the clauses in a path formula contributes more to the treeSAT formula's value than satisfying the same number of clauses scattered over more than one path formula.

Experiments

The performance of randevalssat for computing the value of SSat instances was tested on 27 sets of 100 random formulas, generated under the fixed-clause model using a modified version of `makewff` that guarantees a formula with exactly n variables. For each problem a Sat instance (all existential variables), a Majsat instance (all randomized variables), and all possible instances that contain an equal number of existential and randomized variables in alternating blocks of the same size were constructed. For each of these problems, 10 different partial policy trees were created by sampling randomized variable assignments and constructing the tree paths specified by the samples. The number of assignments sampled w ranged from 5 to 4086 on an approximate logarithmic scale; the larger the number of samples, the greater the similarity of the partial tree to the full tree. Performance was measured by comparing the estimate of the value of the partial policy tree to the exact value of the full policy tree (and, hence, the formula).

Figure 4 shows results for randevalssat on problems with 12 variables, 24 clauses, and 3 literals per clause (results for all 27 sets of problems were similar). The graph shows mean squared error in the value estimate as a function of the number of sampled assignments for all problem types. The mean and variance of the squared error decrease as the number of assignment samples increases. We note that the nonzero error in the limiting case, where the number of assignments sampled is sufficient to construct the full policy tree with high probability, is due to the use of randomized local search. If a more time-consuming systematic search of the partial policy tree is employed, the error is driven to zero as the number of sampled assignments increases (Littman, Majercik, & Pitassi 2001).

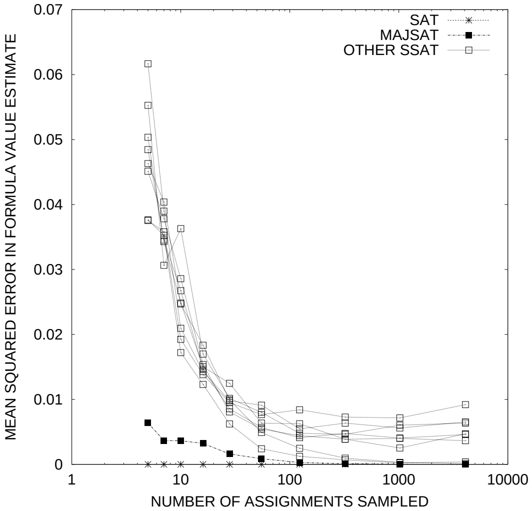


Figure 4: As the number of sampled assignments increases, the accuracy of the randomized local search algorithm increases. Because of local optima in the search space, increasing the number of sampled assignments does not drive the error to zero.

Application to Planning Problems

Once we have translated a probabilistic planning problem into an SSat instance, it would seem to be straightforward to apply the stochastic sampling algorithm to the SSat problem to find an approximation of the optimal contingent plan and an approximation of its probability of success. The situation is complicated, however, by the fact that we use randomized variables to describe observations. This means that a random sample of the randomized variables describes an observation sequence as well as an instantiation of the uncertainty in the domain, and the observation sequence thus produced may not be *observationally consistent*. Informally, an observation sequence is observationally consistent if there exists a sequence of actions and an instantiation of the environment that could possibly produce that observation sequence. For example, assuming perfect sensors, it would be observationally inconsistent to observe at time step t that a valve driver is permanently failed and at time step $t+1$ that it is operational.

Applying the stochastic sampling algorithm directly to planning problems can result in the generation of observationally inconsistent paths in the partial policy tree. And these paths, which are unsatisfied paths regardless of the setting of the action variables chosen, need to be treated differently from observationally consistent paths, which have the potential to become satisfied paths, depending on the values chosen for the decision variables. If the algorithm includes observationally inconsistent paths in its estimate of the probability of success of a given policy, it will tend to underestimate this probability. The algorithm needs to either avoid generating observationally inconsistent paths in the first place, or ignore them in its calculations.

Considerations of efficiency suggest that the first strategy is to be preferred whenever possible, and this suggests two alterations to the stochastic sampling algorithm in order to make it applicable to SSat encodings of planning problems:

- Sample tree paths only from P, the set of paths that are observationally consistent for some action sequence and instantiation of the environment.
- Adjust the evaluation of policy trees as follows: Instantiating the decision variables in the policy tree selects $P' \subseteq P$, the set of paths in P that are observationally consistent for that setting of decision variables and some instantiation of the environment. Let $S \subseteq P'$ be those paths in P' that are satisfied paths. Then, the probability of success of the contingent plan represented by that policy tree is $\frac{|S|}{|P'|}$.

There are various algorithmic issues that need to be addressed. First, we need to find an efficient way of constructing P and P'. One possibility is to use user-supplied information to prune at least some of the observationally inconsistent paths when constructing P. A second possibility is to construct encodings such that unit propagation can be used to efficiently detect observational inconsistencies.

Second, the algorithm returns a partial assignment strategy, or policy, that specifies how each decision variable should be set given the settings of the decision and observation variables that precede it in the quantifier ordering, but this is only specified for those situations represented by paths in the random sample used to construct the partial tree. The algorithm implicitly assumes that performance on missing branches is the same as the average performance over all paths in the partial tree, but doesn't actually find a plan that achieves this performance. One possible strategy for addressing this issue would be an iterative approach that alternates planning and evaluation. In this approach, each iteration would perform an evaluation of the current policy (perhaps by simulating executions of the policy) and use that evaluation to guide the construction of a partial policy tree that would lead to a better policy. This is similar to approaches that identify the most severe problems in an imperfect plan and then attempt to correct them (Drummond & Bresina 1990; Onder & Pollack 1997).

Summary

The stochastic sampling algorithm randevalssat appears to be a promising approximation method for SSat encodings of planning problems. Its primary strength is the use of sampling to convert the problem to a lower complexity problem and its use of randomized local search to solve that problem efficiently. A feature of this process is that it does not necessarily completely discard the probabilities of the original problem (as would, say, a conversion that merely rounded off the probabilities of the randomized variables to 0 and 1 and set their truth values accordingly). It would, for example, be possible to modify the algorithm such that the probabilities of the randomized variables are used to direct the construction of the partial policy tree. A more substantial modification, discussed above, would be to iteratively build the partial policy tree by using the solution of the partial policy tree in a given iteration to construct a better partial policy tree (and solution) in the next iteration.

This algorithm has some weaknesses. First, randevalssat does not return an answer whose correctness is "easy" to certify; this is to be expected, given the complexity of SSat problems. Second, there are SSat problems for which the algorithm needs provably large samples (Littman, Majercik, & Pitassi 2001). Third, randevalssat is subject to the same pitfalls as other randomized local search algorithms; in particular, it may become stuck in local optima. Allowing the algorithm to restart after a period of no progress helps minimize, but does not erase, this problem. Finally, the memory requirements of the algorithm can be prohibitively large. This weakness can be partially overcome by more memory-efficient implementations, but problems that require a large number of samples to produce an accurate answer will inherently generate large treeSAT formulas.

Acknowledgments: The first author acknowledges the support of a NASA GSRP Fellowship.

References

Boutilier, C., and Poole, D. 1996. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1168–1175. AAAI Press/The MIT Press.

Brooks, R. 2000. Personal communication.

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 138–144. Morgan Kaufmann.

Kearns, M. J.; Mansour, Y.; and Ng, A. Y. 1999. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *IJCAI*, 1324–1231.

Littman, M. L.; Majercik, S. M.; and Pitassi, T. 2001. Stochastic Boolean satisfiability. *Journal of Automated Reasoning* 27(3):251—296.

Majercik, S. M., and Littman, M. L. 1999. Contingent planning under uncertainty via stochastic satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 549–556. The AAAI Press/The MIT Press.

Onder, N., and Pollack, M. E. 1997. Contingency selection in plan generation. In *Proceedings of the Fourth European Conference on Planning*.

Papadimitriou, C. H. 1985. Games against nature. *Journal of Computer Systems Science* 31:288–301.