Contingent Planning Under Uncertainty via Stochastic Satisfiability*

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

Abstract

We describe two new probabilistic planning techniques—C-MAXPLAN and ZANDER—that generate contingent plans in probabilistic propositional domains. Both operate by transforming the planning problem into a stochastic satisfiability problem and solving that problem instead. C-MAXPLAN encodes the problem as an E-MAJSAT instance, while ZANDER encodes the problem as an S-SAT instance. Although S-SAT problems are in a higher complexity class than E-MAJSAT problems, the problem encodings produced by ZANDER are substantially more compact and appear to be easier to solve than the corresponding E-MAJSAT encodings. Preliminary results for ZANDER indicate that it is competitive with existing planners on a variety of problems.

Introduction

When planning under uncertainty, any information about the state of the world is precious. A *contingent plan* is one that can make action choices contingent on such information. In this paper, we present an implemented framework for contingent planning under uncertainty using stochastic satisfiability.

Our general motivation for developing the probabilistic-planning-as-stochastic-satisfiability paradigm was to explore the potential for deriving performance gains in probabilistic domains similar to those provided by SATPLAN (Kautz & Selman 1996) in deterministic domains. There are a number of advantages to encoding planning problems as satisfiability problems. First, the expressivity of Boolean satisfiability allows us to construct a very general planning framework. Another advantage echoes the intuition behind reduced instruction set computers; we wish to translate planning problems into satisfiability problems for which we can develop highly optimized solution techniques using a small number of extremely efficient operations. Supporting this goal is the fact that satisfiability is a fundamental problem in computer science and, as such, has been studied intensively. Numerous techniques have been developed to solve satisfiability problems as efficiently as possible. Stochastic satisfiability is less well-studied, but many satisfiability techniques carry over to stochastic satisfiability nearly intact (Littman, Majercik, & Pitassi 1999).

There are disadvantages to this approach. Problems that can be compactly expressed in representations used by other planning techniques often suffer a significant blowup in size when encoded as Boolean satisfiability problems, degrading the planner's performance. Automatically producing maximally efficient plan encodings is a difficult problem. In addition, translating the planning problem into a satisfiability problem obscures the structure of the problem, making it difficult to use our knowledge of and intuition about the planning process to develop search control heuristics or prune plans.

Our planners solve probabilistic propositional planning problems: states are represented as an assignment to a set of Boolean state variables (fluents) and actions map states to states probabilistically. Problems are expressed using a dynamic-belief-network representation. A subset of the state variables are declared *observable*, meaning that a plan can be made contingent on any of these variables. This scheme is sufficiently expressive to allow a domain designer to make a domain fully observable, unobservable, or to have observations depend on actions and states in probabilistic ways.

We describe how to map the problem of contingent planning in a probabilistic propositional domain to two different probabilistic satisfiability problems. C-MAXPLAN, the first approach, encodes the planning problem as an E-MAJSAT instance (Majercik & Littman 1998). A set of Boolean variables (the *choice* variables) encodes the contingent plan and a second set (the *chance* variables) encodes the probabilistic outcome of the plan—the satisfiability problem is to find the setting of the choice variables that maximizes the probability of satisfaction with respect to the chance variables. The efficiency with which the resulting E-MAJSAT problem is solved, however, depends critically on the plan representation. ZANDER, the second approach, encodes the planning problem as an S-SAT instance (Papadimitriou 1985). Here, we intermingle choice variables and chance variables so that values for choice variables encoding actions can be chosen conditionally based on the values of earlier chance variables encoding observations. ZANDER encodings are substantially more compact than C-MAXPLAN encodings, and this appears to more than offset the fact that S-SAT

* To appear in the *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, 1999.

lies in a higher complexity class than E-MAJSAT.

In the remainder of this section, we describe our domain representation and the stochastic satisfiability framework. In the following section, we describe C-MAXPLAN, providing evidence that its performance is quite sensitive to the plan representation. The next two sections introduce the S-SAT-based ZANDER encoding and an algorithm for solving S-SAT instances to find the optimal plan. The section after that reports on a set of comparative experiments; even with our preliminary S-SAT solver, ZANDER appears to be competitive with existing planners across a variety of planning problems. We conclude with some ideas for further work.

Probabilistic Planning Representation

The contingent planners we developed work on partially observable probabilistic propositional planning domains. Such a domain consists of a finite set P of n distinct *propositions*, any 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. A proposition p whose initial assignment is independent of all other propositions has a tree consisting of a single node labeled by the probability with which p will be `True` at time 0. A proposition q whose initial assignment is not independent has a decision tree whose nodes are labeled by the propositions that q depends on and whose leaves specify the probability with which q will be `True` at time 0. *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 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 observable proposition has, as its basis, a proposition that represents the actual status of the thing being observed. (Note that although values are assigned to observable propositions in the initial state, no action at time 1 makes use of these propositions in its decision trees, since there are no valid observations at time 0.)

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.

For example, consider a simple domain based on the TIGER problem of Kaelbling, Littman, & Cassandra (1998). The domain consists of four propositions: tiger-behind-left-door, dead, rewarded and hear-tiger-behind-left-door, the last of which is observable. In the initial state, tiger-behind-left-door is `True` with probability 0.5, dead is `False`, rewarded is `False`, and hear-tiger-behind-left-door is `False` (although irrelevant). The goal states are specified by the partial assignment (rewarded, (not dead)). The three actions are **listen-for-tiger**, **open-left-door**, and **open-right-door** (Figure 1). Actions **open-left-door** and **open-right-door** make reward `True`, as long as the tiger is not behind that door (we assume the tiger is behind the right door if tiger-behind-left-door is `False`). Since tiger-behind-left-door is not observable, the **listen** action becomes important; it causes the observable hear-tiger-behind-left-door proposition to become equal to tiger-behind-left-door with probability 0.85 (and its negation otherwise). By **listen**ing multiple times, it becomes possible to determine the likely location of the tiger.

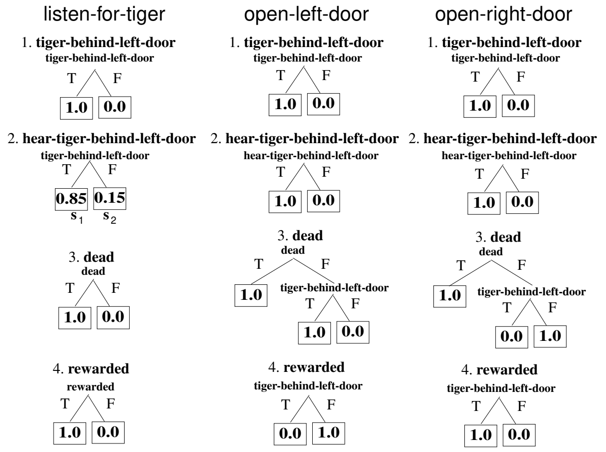


Figure 1: The effects of the actions in the TIGER problem are represented by a set of decision trees.

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`. Fixed-horizon deterministic planning problems can be encoded by SAT formulas (Kautz & Selman 1996).

A formal definition of the SAT decision problem follows. Let $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle$ be a collection of n Boolean variables, and $\phi(\mathbf{x})$ be a CNF Boolean formula on these variables with m clauses. For example, $(x_1 + \overline{x_2} + x_4)(x_2 + x_3 + x_4)(\overline{x_1} + \overline{x_2} + \overline{x_3})$ is a CNF formula with $n = 4$ variables and $m = 3$ clauses. This paper uses "1" and "0" for `True` and `False`, multiplication for conjunction, and addition for disjunction. Logical negation is defined as $\overline{x} = 1 - x$. With respect to a formula $\phi(\mathbf{x})$, an assignment to the Boolean variables $x_1, \ldots, x_n$ is *satisfying* if $\phi(\mathbf{x}) = 1$. In other words, a satisfying assignment makes the formula `True`. The decision problem SAT asks whether a given Boolean formula $\phi(\mathbf{x})$ in CNF has a satisfying assignment $(\exists x_1, \ldots, \exists x_n(\phi(\mathbf{x}) = 1))$.

Papadimitriou (1985) explored an extension of SAT in which a random quantifier is introduced. The stochastic SAT (S-SAT) problem is to evaluate a Boolean formula in which existential and random quantifiers alternate:

$$\exists x_1, \text{Я} x_2, \exists x_3, \ldots, \exists x_{n-1}, \text{Я} x_n (E[\phi(\mathbf{x})] \geq \theta).$$

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 of the Boolean formula $\phi(\mathbf{x})$ is at least a threshold θ. This type of satisfiability consists of alternating between making a choice of value for an odd-numbered variable with a chance selection of a value for an even-numbered variable; hence, Papadimitriou referred to S-SAT as a "game against nature." In our S-SAT problems, we will allow *blocks* of existential and random quantifiers to alternate. Furthermore, we will allow *annotated random quantifiers* such as $\text{Я}^{0.2}$, which takes on value `True` with probability 0.2 and `False` with probability 0.8. S-SAT, like the closely related quantified Boolean formula problem, is PSPACE-complete. The specification of an S-SAT problem consists of the Boolean formula $\phi(\mathbf{x})$, the probability threshold θ, and the ordering of the quantifiers.

Different thresholds and patterns of quantifiers in S-SAT instances result in different computational problems, complete for different complexity classes. An S-SAT problem with a threshold of 1.0 and a single block of existentially quantified variables is equivalent to the NP-complete problem SAT. An S-SAT problem with an arbitrary threshold and a single block of existentially quantified variables followed by a single block of randomly quantified variables is equivalent to the $\mathrm{NP}^{\mathrm{PP}}$-complete problem E-MAJSAT. As we will describe, both E-MAJSAT formulas and S-SAT formulas can be used to encode planning problems.

Related Work

The type of partially observable planning problem we address, featuring actions with probabilistic effects and noisy observations, is a form of partially observable Markov decision process (POMDP). Algorithms that use a *flat* representation for POMDPs have been around for many years. In this section, we focus on more recent algorithms that exploit propositional state representations. Of course, any algorithm that can solve a planning problem in a flat representation can also be used to solve a problem in the propositional representation by enumerating states; in fact, this approach is often the fastest for domains with up to five or six fluents.

One of the most well-known contingent planners for probabilistic domains is C-BURIDAN (Draper, Hanks, & Weld 1994), which uses tree-based, probabilistic STRIPS operators to extend partial-order planning to stochastic domains. C-BURIDAN searches for a type of contingent plan whose probability of success meets or exceeds some prespecified threshold. As Onder & Pollack (1997) point out, however, there are some problems with C-BURIDAN, and these could prevent it from solving arbitrary partially observable planning problems. MAHINUR (Onder & Pollack 1997) is a probabilistic partial-order planner that corrects these deficiencies by combining C-BURIDAN's probabilistic action representation and system for managing these actions with a CNLP-style approach to handling contingencies. The novel feature of MAHINUR is that it identifies those contingencies whose failure would have the greatest negative impact on the plan's success and focuses its planning efforts on generating plan branches to deal with those contingencies. Onder & Pollack (1997) identify several domain assumptions (including a type of subgoal decomposability) that underlie the design of MAHINUR, and there are no guarantees on the correctness of MAHINUR for domains in which these assumptions are violated. Our contingent planners, C-MAXPLAN and ZANDER, correct the deficiencies noted by Onder and Pollack and, in addition, avoid the assumptions made my MAHINUR, thus resulting in planners that are applicable to more general domains.

CONFORMANT GRAPHPLAN (Smith & Weld 1998) deals with uncertainty in initial conditions and action outcomes by attempting to construct a non-sensing, noncontingent plan that will succeed in all cases. PGRAPHPLAN (Blum & Langford 1998) employs forward search through the planning graph to find the contingent plan with the highest expected utility in a completely observable stochastic environment. SENSORY GRAPHPLAN (SGP) (Weld, Anderson, & Smith 1998), constructs plans with sensing actions that gather information to be used later in distinguishing between different plan branches. Thus, SGP is an approach to constructing contingent plans. However, SGP has not been extended to handle actions with uncertain effects (except in the conformant case) and imperfect observations, so it is only applicable to a subset of partially observable planning problems.

Boutilier & Poole (1996) describe an algorithm for solving partially observable planning problems based on an earlier algorithm for completely observable problems. While promising, little computational experience with this algorithm is available.

Our planners, C-MAXPLAN and ZANDER, are based on earlier work on MAXPLAN (Majercik & Littman 1998), a planner for unobservable domains. Both are based on stochastic satisfiability and can handle general (finite-horizon) partially observable planning problems. They allow both states and observations to be in a compact propositional form, and so can be used to attack large-scale planning problems.

For the partially observable planning problems they can solve, MAHINUR and SGP appear to run at state-of-the-art speeds; in the Results section, we will report favorable comparisons of ZANDER with MAHINUR and SGP, as well as an implementation of a POMDP algorithm for flat domains, on some standard test problems.

C-MAXPLAN

MAXPLAN (Majercik & Littman 1998) was initially developed to solve probabilistic planning problems in completely unobservable domains. MAXPLAN works by first converting the planning problem to an E-MAJSAT problem, which is an S-SAT problem with a single block of existential (choice) variables followed by a single block of random (chance) variables. The choice variables encode candidate plans, and the chance variables encode the probabilistic outcome of the plan. MAXPLAN solves the E-MAJSAT problem using a modified version of the Davis-Putnam-Logemann-Loveland (DPLL) procedure for determining satisfiability. Essentially, it uses DPLL to determine all possible satisfying assignments, sums the probabilities of the satisfying assignments for each possible choice-variable assignment, and then returns the choice-variable assignment (plan) with the highest probability of producing a satisfying assignment (goal satisfaction). The algorithm uses an efficient splitting heuristic (*time-ordered splitting*) and *memoization* (Majercik & Littman 1998) to accelerate this procedure. Note that in MAXPLAN an n-step plan is encoded as a selection of one action out of the $|A|$ possible actions for each of the n steps. This is shown graphically in Figure 2(a) for a 3-step plan, where action selection is indicated by bold circles.

The MAXPLAN approach can also handle contingent planning problems if given an appropriate problem encoding. A generic E-MAJSAT encoding for contingent plans follows, where c_1 is the number of choice variables needed to specify the plan, c_2 is the number of state variables (one for each proposition at each time step), and c_3 is the number of chance variables (one for each possible stochastic outcome at each time step):

$$\overbrace{\exists x_1, \ldots, \exists x_{c_1}}^{\text{the plan}} \overbrace{\exists y_1, \ldots, \exists y_{c_2}}^{\text{the state}} \overbrace{\rotatebox{180}{R}^{\rho_1} z_1, \ldots, \rotatebox{180}{R}^{\rho_{c_3}} z_{c_3}}^{\text{random outcomes}}$$
$$(E[(\text{initial/goal conditions } (y,z)\text{-clauses})$$
$$(\text{action exclusion } (x)\text{-clauses})$$
$$(\text{action outcome } (x,y,z)\text{-clauses})] \geq \theta).$$

The formula picks the plan and the sequence of states encountered, and then randomly selects the outcome of actions.[1] The clauses insist that initial and goal conditions are satisfied, one action is selected per time step, and that the sequence of states selected is valid given the selected actions and random outcomes.

More specifically, a contingent action in contingent MAXPLAN (C-MAXPLAN) is expressed as a group of actions, all of which execute, but only one of which has an impact on the state (the one whose set of conditions matches the set of observations generated by previous actions). Since the condition sets of the actions are mutually exclusive, the net result is that at most one action in the group will *effectively* execute (*i.e.* affect the state), depending on current conditions. Thus, in C-MAXPLAN, it is more appropriate to refer to *action* steps than time steps. The difference between MAXPLAN encodings and C-MAXPLAN encodings is shown graphically in Figure 2. Figure 2(a) shows a 3-step plan in MAXPLAN, where selected actions are indicated by bold circles. Figure 2(b) shows a 3-action plan in C-MAXPLAN. Actions are still selected as in MAXPLAN, but now all actions, except for Action 1, have conditions attached to them (the c variables in the boxes above the action selection boxes). These conditions specify when the action will effectively execute. In Figure 2(b), Action 2 will effectively execute if condition c_1 is `True` (bold circle) and condition c_2 is `False` (bold circle with slash). Condition c_3 is indicated to be irrelevant (it can be `True` *or* `False`) by a broken circle. Action 3 will effectively execute if condition c_1 is `False` and condition c_2 is `True` (condition c_3 is, again, irrelevant).

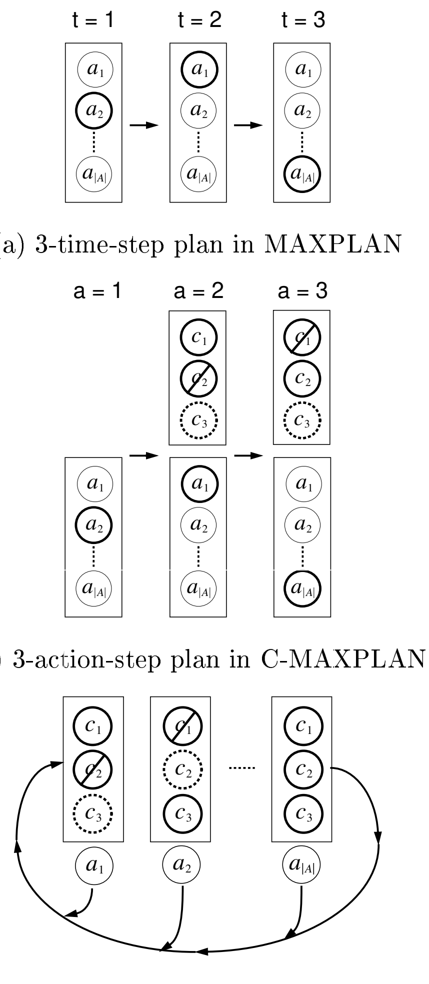
To encode contingent plans in this manner, we need additional variables and clauses, and we need to alter the decision trees of the actions (which will alter some of the clauses as well). The key clauses in the contingent plan encodings are those clauses that model the satisfaction of conditions. At a high level, these clauses enforce the notion that if if condition c specifies that proposition p have truth status T and the variable indicating that our current observation of p is valid is `True`, and the variable indicating our perception of p has truth status T, *then* c is satisfied.

Initial tests of this technique were promising; the most basic version of C-MAXPLAN solved a contingent version of the SAND-CASTLE-67 problem, the SHIP-REJECT problem (Draper, Hanks, & Weld 1994), and the MEDICAL-1ILL problem (Weld, Anderson, & Smith 1998) in 3.5, 5.25, and 0.5 cpu seconds, respectively on a 300 MHz UltraSparcIIi. But tests on the MEDICAL-4ILL problem (Weld, Anderson, & Smith 1998) were disappointing; even accelerated versions of C-MAXPLAN had not solved the problem after several *days*.

We obtained significantly better performance by implementing three improvements. First, instead of searching for the optimal plan of a given length, we search for an optimal small *policy* to be applied for a given number of steps. In this approach, the decision trees from all actions for each proposition p are merged into a single decision tree that describes the impact of *all* the actions on p via a cascade of condition-fulfillment variables. Essentially, the decision tree says: "If the conditions specified by the policy for action a are satisfied, then decide the status of p according to a's decision tree; otherwise, if the conditions for action b are satisfied, then decide the status of p according to b's decision tree; ...; otherwise, the status of p remains the same."

In this encoding, we have a single action—*follow-the-policy*—and the choice variables are used to describe that policy. A policy is specified by describing the conditions under which each *primitive* action (an ac-

[1] In fact, in our implementation, the random outcome variables precede the state variables. Although the resulting encoding isn't precisely E-MAJSAT, the values of the state variables are forced given the outcome variables, and so their quantifiers are not significant.



(a) 3-time-step plan in MAXPLAN

(b) 3-action-step plan in C-MAXPLAN

(c) Policy encoding in C-MAXPLAN

Figure 2: MAXPLAN and the two styles of encoding in C-MAXPLAN encode plans in different ways.

tion in the original domain) should be executed. Figure 2(c) shows a policy: conditions (in the boxes) are specified for each action, and one cycle of policy execution executes the first action whose conditions match the current state. In this formulation of the problem, the algorithm searches for the setting of these policy-specification variables that maximizes the probability of satisfying the E-MAJSAT formula (achieving the goals).

The primary advantage of this approach appears to be a more compact encoding of the problem, achieved by exploiting the fact that the status of a given proposition can typically be changed by only a small percentage of the actions in the domain. (This is similar to the use of *explanatory frame axioms* by Kautz, McAllester, & Selman (1996) to reduce the size of their *linear* SAT encodings of planning problems.)

Second, we adapted the DPLL splitting heuristic described by Bayardo & Schrag (1997) for use in C-MAXPLAN. This heuristic selects an initial pool of candidates based on a score that rewards variables that appear both negated and not negated in binary clauses. This initial pool is rescored based on the number of unit propagations that occur for each assignment to each variable, rewarding variables for which both truth values induce unit propagations. Essentially, this heuristic tries to find a variable that will induce the highest number of unit propagations, thereby maximizing pruning.

Third, we implemented a *thresholding* technique similar to that of C-BURIDAN and MAHINUR. If, instead of insisting on finding the plan with optimal probability, we supply a minimum desired probability, we can prune plans based on this threshold. For a choice variable, if the probability of success given an assignment of `True` is higher than our threshold, we can prune plans in which this variable would be assigned `False`. For a chance variable, we can perform a similar kind of pruning (although the thresholds passed down the tree must be appropriately adjusted). But, for chance variables, if the probability of success given an assignment of `True` is *low* enough, we can determine that the probability weighted average of both truth assignments will not meet our adjusted threshold and can return failure immediately (Littman, Majercik, & Pitassi 1999).

With these improvements, C-MAXPLAN can solve the MEDICAL-4ILL problem in approximately 100 cpu seconds. But, there are issues that make this approach problematic. First, the results described above indicate that the performance of this approach is very sensitive to the details of the plan encoding, making it less robust than desired. Second, if two actions could be triggered by the same set of conditions, only the first one in the decision-tree cascade will be triggered, so the construction of the decision tree introduces unwanted bias. Finally, plan encodings for problems in which actions need to be conditioned on an entire *history* of observations grow exponentially with the length of the history.

$$\overbrace{\exists x_{1,1},\ldots,\exists x_{1,c_1}}^{\text{first action}}\overbrace{\text{ᴚ} w_{1,1},\ldots,\text{ᴚ} w_{1,c_2}}^{\text{first observation}}\cdots\overbrace{\text{ᴚ} w_{n-1,1},\ldots,\text{ᴚ} w_{n-1,c_2}}^{\text{last observation}}\overbrace{\exists x_{n,1},\ldots,\exists x_{n,c_1}}^{\text{last action}}\overbrace{\text{ᴚ}^{\rho_1} z_1,\ldots,\text{ᴚ}^{\rho_{c_4}} z_{c_4}}^{\text{random outcomes}}\overbrace{\exists y_1,\ldots,\exists y_{c_3}}^{\text{the state}}$$

$$(E[(\text{initial/goal conditions } (y,z)\text{-clauses})(\text{action exclusion } (x)\text{-clauses})(\text{action outcome } (w,x,y,z)\text{-clauses})] \geq \theta).$$

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

Figure 3: Contingent planning problems can be encoded as an instance of S-SAT.

ZANDER

In an S-SAT 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 another way of mapping contingent planning problems to stochastic satisfiability: encode the contingent plan in the variable ordering associated with the S-SAT formula. By alternating blocks of existential variables that encode actions and blocks of random 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 S-SAT encoding for contingent plans appears in Figure 3. This approach is agnostic as to the structure of the plan; the type of plan returned is algorithm dependent. Our S-SAT solver, described below, constructs tree-structured proofs; these correspond to tree-structured plans that contain a branch for each observable variable. Other solvers could produce DAG-structured, subroutine-structured, or value-function-based plans.

The quantifiers naturally 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 choice-variable blocks (one for each action choice) and chance-variable blocks (one for each set of possible observations at a time step).[2] In our TIGER problem, each action variable block would be composed of the three possible actions—**listen-for-tiger**, **open-left-door**, and **open-right-door**—and each observation variable block would be composed of the single variable hear-tiger-behind-left-door.

The domain uncertainty segment is a single block containing all the chance 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. In the TIGER problem, there would be a chance variable (probability = 0.85) associated with the outcome of each **listen-for-tiger** action.

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.

Algorithm Description

C-MAXPLAN finds the assignment to the choice variables that maximizes the probability of getting a satisfying assignment with respect to the chance variables. ZANDER, however, must find an assignment *tree* that specifies the optimal choice-variable assignment given all possible settings of the observation variables. Note that we are no longer limiting the size of the plan to be polynomial in the size of the problem; the assignment tree can be exponential in the size of the problem.

The most basic variant of the solver follows the variable ordering exactly, constructing a binary DPLL tree of all possible assignments. Figure 4 depicts such a tree; each node 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. The tree in Figure 4 shows the first seven variables in the ordering for the 2-step TIGER problem: the three choice variables encoding the action at time-step 1, the single observation chance variable, and the three choice variables encoding the action at time-step 2 (triangles indicate subtrees for which details are not shown). The observation variable is a branch point; the optimal assignment to the remaining variables will, in general, be different for different values of this variable.

This representation of the planning problem is sim-

[2] Although an observation is associated with a chance variable, it marks a branch point in the plan, and we want the result of an observation to be the *sum*, not the probability weighted average, of the probabilities associated with the two possible truth settings of the chance variable. We accomplish this by setting the probability associated with an observation chance variable to 0.5 and adjusting the resulting plan probability upward by the same factor.

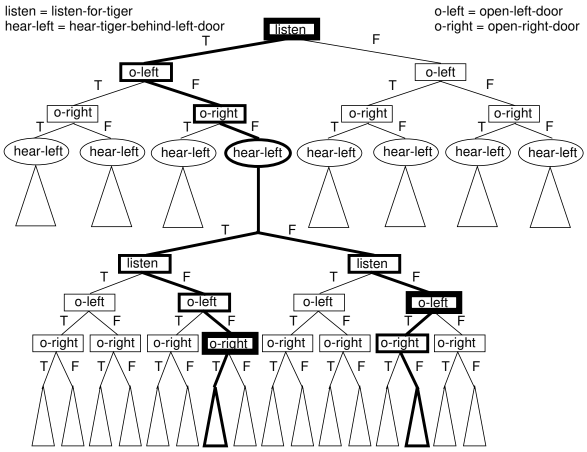


Figure 4: ZANDER selects an optimal subtree.

ilar to *AND/OR trees* and *MINIMAX trees* (Nilsson 1980). Choice variable nodes are analogous to OR, or MAX, nodes, and chance variable nodes are analogous to AND, or MIN, nodes. But the probabilities associated with chance variables (our opponent is nature) make the analogy somewhat inexact. Our trees are more similar to the *MINIMAX trees with chance nodes* described by Ballard (1983) but without the MIN nodes—instead of a sequence of alternating moves by opposing players mediated by random events, our trees represent a sequence of moves by a single player mediated by the randomness in the planning domain.

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 a choice 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 chance 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 probability of success. In Figure 4, the plan portion of this subtree appears in bold, with action choices (action variables set to `True`) in extra bold. The optimal plan is: **listen-for-tiger**; if hear-tiger-behind-left-door is `True`, **open-right-door**; if `False`, **open-left-door**.

We use three pruning techniques to avoid checking every possible truth assignment. Whenever a choice or chance variable appears alone in an active clause, *unit propagation* assigns the forced value to that variable. This is valid since, even if we postponed the assignment until we reached that variable in the quantifier ordering, we would still need to assign the forced value. Whenever a choice variable appears always negated or always not negated in all active clauses, *variable purification* assigns the appropriate value to that variable. This is valid since the variable would still be pure even if we postponed the assignment until we reached that variable in the quantifier ordering. *Thresholding*, as described earlier, allows us to prune plans based on a prespecified threshold probability of success, and is similar to the MINIMAX tree **cutoffs* described by Ballard (1983).

Like the solutions found by ZANDER, the solution of an AND/OR tree is a subtree satisfying certain conditions. Algorithms for solving these trees, such as AO* (Nilsson 1980), try to combine the advantages of dynamic programming (reuse of common subproblems) with advantages of branch-and-bound (use of heuristic estimates to speed up the search process). These algorithms operate by repeating a 2-phase operation: use heuristic estimates to identify the next node to expand, then use dynamic programming to re-evaluate all nodes in the current subgraph. In contrast to this approach, which must follow a prescribed variable ordering, ZANDER can consider variables out of the order prescribed by the problem, when this allows it to prune subtrees (as in unit propagation and variable purification). A worthwhile area of research would be to compare the performance of these two approaches and attempt to develop techniques that combine the advantages of both.

Results

We tested three variants of ZANDER on problems drawn from the planning literature (see Figure 5). All tests were done on a 300 MHz UltraSparcIIi. The basic solver, which uses only variable splitting and, essentially, checks every possible assignment (SPLITTING), the basic solver augmented with unit propagation and purification (UNITPURE), and the basic solver with unit propagation, purification, and thresholding (THRESH).

The TIGER problems (with horizon increasing from one to four) contain uncertain initial conditions and a noisy observation. Note that in the 4-step TIGER problem, the agent needs the entire observation history in order to act correctly. The SHIP-REJECT problem has the same characteristics as the TIGER problem, along with a causal action (**paint**) that succeeds only part of the time. In the MEDICAL-4ILL problem, we have uncertain initial conditions, multiple perfect observations, and causal actions with no uncertainty. The EXTENDED-PAINT problems (Onder 1998) have no uncertainty in the initial conditions, but require that probabilistic actions be interleaved with perfect observations. Finally, the COFFEE-ROBOT problem, similar to a problem described by Boutilier & Poole (1996), is a larger problem (7 actions, 2 observation variables, and 8 state propositions in each of 6 time steps) with uncertain initial conditions, but perfect causal actions and observations.

As expected, the performance of SPLITTING is poor except on the simplest problems. But, the results for UNITPURE and THRESH are very encouraging; the techniques used in these variants are able to reduce solution times by as much as 5 orders of magnitude. These two variants of ZANDER appear to be quite competitive with other planners; the tests we have conducted so

Problem	Threshold Probability of Success	Solution Time (cpu seconds)		
		SPLITTING	UNITPURE	THRESH
TIGER-1	0.5	0.02	0.02	0.01
TIGER-2	0.85	0.12	0.02	0.02
TIGER-3	0.85	2.81	0.05	0.04
TIGER-4	0.93925	72.19	0.19	0.08
SHIP-REJECT	0.9215	25.40	0.06	0.06
MEDICAL-4ILL	1.0	196.40	1.77	0.25
EXTPAINT-4	0.3125	12,606.47	0.44	0.13
EXTPAINT-7	0.773437	NA	164.96	31.35
COFFEE-ROBOT	1.0	46,152.25	1,827.67	769.20

Figure 5: Unit propagation, purification, and thresholding can improve performance greatly.

far, while not exhaustive, are encouraging. UNITPURE and THRESH solve the TIGER-4 problem in 0.19 and 0.08 cpu seconds respectively, compared to 0.04 cpu seconds for "Lark" pruning (Kaelbling, Littman, & Cassandra 1998) on the corresponding finite-horizon POMDP. These ZANDER variants can solve the MEDICAL-4ILL problem in 1.77 and 0.25 cpu seconds respectively, compared to 44.54 cpu seconds for SGP. And both variants can solve the SHIP-REJECT problem in 0.06 cpu seconds compared to 0.12 cpu seconds for MAHINUR.

Further Work

ZANDER's more straightforward problem encodings and better performance make it a more promising candidate for further work than C-MAXPLAN. There are a number of possibilities for improvements. Currently, ZANDER separately explores and saves two plan execution histories that diverge and remerge, constructing a plan tree when a directed acyclic graph would be more efficient. We would like to be able to memoize subplan results (a technique used by MAXPLAN) so that when we encounter previously solved subproblems, we can merge the current plan execution history with the old history.

We would like ZANDER to evaluate plans using a broader conception of *utility* than probability of success alone. For example, ZANDER sometimes returns an unnecessarily large plan; we would like the planner to discriminate between plans with equal probability of success using length as a criterion.

Better splitting heuristics could boost performance. Although we are constrained by the prescribed quantifier ordering, a splitting heuristic can be used within a block of similarly quantified variables. Early experiments indicate this can improve performance in bigger problems, where such blocks are large (Littman, Majercik, & Pitassi 1999).

We would like to create approximation techniques for solving larger planning problems. One possibility, currently being developed, uses random sampling to limit the size of the contingent plans we consider and stochastic local search to find the best size-bounded plan. This approach has the potential to quickly generate a suboptimal plan and then, in the remaining available planning time, adjust this plan to improve its probability of success. This sacrifice of optimality for "anytime" planning with performance bounds may not improve worst-case complexity, but it is likely to help for typical problems.

Finally, we would like to explore the possibility of using the approximation technique we are developing in a framework that interleaves planning and execution. This could improve efficiency greatly (at the expense of optimality) by making it unnecessary to generate a plan that considers all contingencies.

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

References

Ballard, B. W. 1983. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence* 21(3):327—350.

Bayardo, Jr., R. J., and Schrag, R. C. 1997. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 203–208. AAAI Press/The MIT Press.

Blum, A. L., and Langford, J. C. 1998. Probabilistic planning in the Graphplan framework. In *Working Notes of the Workshop on Planning as Combinatorial Search, held in conjunction with the Fourth International Conference on Artificial Intelligence Planning.*

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.

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, 76–82.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1–2):99–134.

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.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR-96)*, 374–384.

Littman, M. L.; Majercik, S. M.; and Pitassi, T. 1999. Stochastic boolean satisfiability. Submitted.

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.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga Publishing Company.

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

Onder, N. 1998. Personal communication.

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

Smith, D. E., and Weld, D. S. 1998. Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 889–896. AAAI Press/The MIT Press.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending Graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 897–904. AAAI Press/The MIT Press.