APPSSAT: Approximate Probabilistic Planning Using Stochastic Satisfiability

Stephen M. Majercik

Bowdoin College, Brunswick ME 04011, USA,
smajerci@bowdoin.edu,
http://www.bowdoin.edu/~smajerci

Abstract. We describe APPSSAT, an anytime probabilistic contingent planner based on ZANDER, a probabilistic contingent planner that operates by converting the planning problem to a stochastic satisfiability (SSAT) problem and solving that problem instead [1]. The values of some of the variables in an SSAT instance are probabilistically determined; APPSSAT considers the most likely instantiations of these variables (the most probable situations facing the agent) and attempts to construct an approximation of the optimal plan that succeeds under those circumstances, improving that plan as time permits. Given more time, less likely instantiations/situations are considered and the plan is revised as necessary. In some cases, a plan constructed to address a relatively low percentage of possible situations will succeed for situations not explicitly considered as well, and may return an optimal or near-optimal plan. This means that APPSSAT can sometimes find optimal plans faster than ZANDER. And the anytime quality of APPSSAT means that suboptimal plans could be efficiently derived in larger time-critical domains in which ZANDER might not have sufficient time to calculate the optimal plan. We describe some preliminary experimental results and suggest further work needed to bring APPSSAT closer to attacking real-world problems.

1 Introduction

Previous research has extended the planning-as-satisfiability paradigm to support probabilistic contingent planning; in [1], it was shown that a probabilistic, partially observable, finite-horizon, contingent planning problem can be encoded as a stochastic satisfiability (SSAT) [2] instance such that the solution to the SSAT instance yields a contingent plan with the highest probability of reaching a goal state. This has been used to construct ZANDER, a competitive probabilistic contingent planner [1]. APPSSAT is a probabilistic contingent planner based on ZANDER that produces an approximate contingent plan and improves that plan as time permits. APPSSAT does this by considering the most probable situations facing the agent and constructing a plan, if possible, that succeeds under those circumstances. Given more time, less likely situations are considered and the plan is revised as necessary.

Other researchers have explored the possibility of using approximation to speed the planning process. In "anytime synthetic projection" a set of control rules establishes a base plan which has a certain probability of achieving the goal [3]. Time permitting, the probability of achieving the goal is incrementally increased by identifying failure situations that are likely to be encountered by the current plan and synthesizing additional control rules to handle these situations. Similarly, MAHINUR is a probabilistic partial-order planner that also creates a base plan with some probability of success and then improves that plan [4].

Exploring approximation techniques in Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs) has been a very active area of research in recent years. In [5] value functions are represented using decision trees and these decision trees are pruned so that the leaves represent ranges of values, thereby approximating the value function. Evidence that the value function of a factored MDP can often be well approximated using a factored value function has been presented in [6], and it is shown that this approximation technique can be used as a subroutine in a policy iteration process to solve factored MDPs [7]. A method for choosing, with high probability, approximately optimal actions in an infinite-horizon discounted Markov decision process using truncated action sequences and random sampling is described in [8]. In [9] the authors transform a POMDP into a simpler *region observable* POMDP in which it is assumed an oracle tells the agent what region its current state is in. This POMDP is easier to solve and they use its solution to construct an approximate solution for the original POMDP.

In Section 2, we describe stochastic satisfiability, the basis for both ZANDER and APPSSAT. In Section 3, we describe how ZANDER uses stochastic satisfiability to solve probabilistic planning problems. In Section 4, we describe the APPSSAT algorithm for approximate planning and in Section 5 we describe some preliminary experimental results. We conclude with a discussion of further work.

2 Stochastic Satisfiability

SSAT, suggested in [10] and explored further in [2], is a generalization of satisfiability (SAT) that is similar to quantified Boolean satisfiability (QBF). The ordered variables of the Boolean formula in an SSAT problem, instead of being existentially or universally quantified, are existentially or *randomly* quantified. Randomly quantified variables are `true` with a certain probability, and an SSAT instance is satisfiable with some probability that depends on the ordering of and interplay between the existential and randomized variables. The goal is to choose values for the existentially quantified variables that maximize the probability of satisfying the formula.

More formally, an SSAT problem $\Phi = Q_1 v_1 \ldots Q_n v_n \phi$ is specified by 1) a *prefix* $Q_1 v_1 \ldots Q_n v_n$ that orders a set of n Boolean variables $V = \{v_1, \ldots, v_n\}$ and specifies the quantifier Q_i associated with each variable v_i, and 2) a *matrix* ϕ that is a Boolean formula constructed from these variables. More specifically, the prefix $Q_1 v_1 \dots Q_n v_n$ associates a quantifier Q_i, either existential ($\exists_i$) or randomized ($\rotatebox[origin=c]{180}{R}_i^{\pi_i}$), with the variable v_i. The value of an existentially quantified variable can be set arbitrarily by a solver, but the value of a randomly quantified variable is determined stochastically by π_i, an arbitrary rational probability that specifies the probability that v_i will be `true`. (In the basic SSAT problem described in [2], every randomized variable is `true` with probability 0.5, but it is noted that the probabilities associated with randomized variables can be arbitrary rational numbers.) In this paper, we will use $x_1, x_2, \dots$ for existentially quantified variables and $y_1, y_2, \dots$ for randomly quantified variables.

The matrix ϕ is assumed to be in conjunctive normal form (CNF), i.e. a set of m conjuncted clauses, where each clause is a set of distinct disjuncted literals. A *literal* l is either a variable v (a *positive* literal) or its negation $-v$ (a *negative* literal). For a literal l, $|l|$ is the variable v underlying that literal and $\bar{l}$ is the "opposite" of l, i.e. if l is v, $\bar{l}$ is $-v$; if l is $-v$, $\bar{l}$ is v; A literal l is `true` if it is positive and $|l|$ has the value `true`, or if it is negative and $|l|$ has the value `false`. A literal is *existential* (*randomized*) if $|l|$ is existentially (randomly) quantified. The probability that a randomly quantified variable v has the value `true` (`false`) is denoted $Pr[v]$ ($Pr[-v]$). The probability that a randomized literal l is `true` is denoted $Pr[l]$. As in a SAT problem, a clause is satisfied if at least one literal is `true`, and unsatisfied, or *empty*, if all its literals are `false`. The formula is satisfied if all its clauses are satisfied.

The solution of an SSAT instance is an assignment of truth values to the existentially quantified variables that yields the maximum probability of satisfaction, denoted $Pr[\Phi]$. Since the values of existentially quantified variables can be made contingent on the values of randomly quantified variables that appear earlier in the prefix, the solution is, in general, a *tree* that specifies the optimal assignment to each existentially quantified variable x_i for each possible instantiation of the randomly quantified variables that precede x_i in the prefix. A simple example will help clarify this idea before we define $Pr[\Phi]$ formally. Suppose we have the following SSAT problem:

$$\exists x_1, \rotatebox[origin=c]{180}{R}^{0.7} y_1, \exists x_2 \{\{x_1, y_1\}, \{x_1, \overline{y_1}\}, \{y_1, x_2\}, \{\overline{y_1}, \overline{x_2}\}\} \ . \tag{1}$$

The form of the solution is a noncontingent assignment for x_1 plus two contingent assignments for x_2, one for the case when y_1 is `true` and one for the case when y_1 is `false`. In this problem, x_1 should be set to `true` (if x_1 is `false`, the first two clauses become $\{\{y_1\}, \{\overline{y_1}\}\}$, which specify that y_1 must be both `true` *and* `false`), and x_2 should be set to `true` (`false`) if y_1 is `false`(`true`). Since it is possible to satisfy the formula for both values of y_1, $Pr[\Phi] = 1.0$. If we add the clause $\{\overline{y_1}, x_2\}$ to this instance, however, the maximum probability of satisfaction drops to 0.3: x_1 should still be set to `true`, and when y_1 is `false`, x_2 should still be set to `true`. When y_1 is `true`, however, we have the clauses $\{\{\overline{x_2}\}, \{x_2\}\}$, which insist on contradictory values for x_2. Hence, it is possible to satisfy the formula only when y_1 is `false`, and, since $Pr[-y_1] = 0.3$, the probability of satisfaction, $Pr[\Phi]$, is 0.3.

We will need the following additional notation to define $Pr[\Phi]$ formally. A partial assignment α of the variables V is a sequence of $k \leq n$ literals $l_1; l_2; \dots; l_k$

such that no two literals in α have the same underlying variable. Given l_i and l_j in an assignment α, $i < j$ implies that the assignment to $|l_i|$ was made before the assignment to $|l_j|$. A positive (negative) literal v $(-v)$ in an assignment α indicates that the variable v has the value `true` (`false`). The notation $\Phi(\alpha)$ denotes the SSAT problem Φ' remaining when the partial assignment α has been applied to Φ (i.e. clauses with `true` literals have been removed from the matrix, `false` literals have been removed from the remaining clauses in the matrix, and all variables and associated quantifiers not in the remaining clauses have been removed from the prefix) and $\phi(\alpha)$ denotes ϕ', the matrix remaining when α has been applied. Similarly, given a set of literals L, such that no two literals in L have the same underlying variable, the notation $\Phi(L)$ denotes the SSAT problem Φ' remaining when the assignments indicated by the literals in L have been applied to Φ, and $\phi(L)$ denotes ϕ', the matrix remaining when the assignments indicated by the literals in L have been applied. A literal $l \notin \alpha$ is *active* if some clause in $\phi(\alpha)$ contains l; otherwise it is *inactive*.

Given an SSAT problem Φ, the maximum probability of satisfaction of Φ, denoted $Pr[\Phi]$, is defined according to the following recursive rules:

1. If ϕ contains an empty clause, $Pr[\Phi] = 0.0$.
2. If ϕ is the empty set of clauses, $Pr[\Phi] = 1.0$.
3. If the leftmost quantifier in the prefix of Φ is existential and the variable thus quantified is v, then $Pr[\Phi] = \max(Pr[\Phi(v)], Pr[\Phi(-v)])$.
4. If the leftmost quantifier in ϕ is randomized and the variable thus quantified is v, then $Pr[\Phi] = (Pr[\Phi(v)] \times Pr[v]) + (Pr[\Phi(-v)] \times Pr[-v])$.

These rules express the intuition that a solver can select the value for an existentially quantified variable that yields the subproblem with the higher probability of satisfaction, whereas a randomly quantified variable forces the solver to take the weighted average of the two possible results.

There are simplifications that allow an algorithm implementing this recursive definition to avoid the often infeasible task of enumerating all possible assignments. Of course, if the empty set of clauses, or an empty clause, is reached before a complete assignment is made, the solver can immediately return 1.0, or 0.0, respectively. Further efficiencies are gained by interrupting the normal left-to-right evaluation of quantifiers to take advantage of *unit* and *pure* literals. A literal l is *unit* if it is the only literal in some clause; in this case, $|l|$ must be assigned the value that makes l `true`. A literal l is *pure* if l is active and $\bar{l}$ is inactive; if l is an existential pure literal, $|l|$ can be set to make l `true` without changing $Pr[\Phi]$. These simplifications modify the rules given above for determining $Pr[\Phi]$, but we omit a restatement of the modified rules, instead describing an algorithm to solve SSAT instances based on the modified rules (Figure 1). Note that both ZANDER and APPSSAT construct and return the optimal solution tree (plan), but we omit the details of solution tree construction in the algorithm description.

```
SolveSSAT ($\Phi$)
  if $\phi$ contains an empty clause: return 0.0;
  if $\phi$ is the empty set of clauses: return 1.0;
  if some l is an existential unit literal:
    return SolveSSAT ($\Phi$(l));
  if some l is a randomized unit literal:
    return SolveSSAT ($\Phi$(l)) * Pr[l];
  if some l is an existential pure literal:
    return SolveSSAT ($\Phi$(l));
  if the leftmost quantifier in $\Phi$ is $\exists$ and its variable is v:
    return max(SolveSSAT ($\Phi$(v)), SolveSSAT ($\Phi$(-v)));
  if the leftmost quantifier in $\Phi$ is $\reflectbox{R}$ and its variable is v:
    return SolveSSAT ($\Phi$(v)) * Pr[v] + SolveSSAT ($\Phi$(-v)) * Pr[-v];
```

Fig. 1. The basic algortihm for solving Ssat instances

3 ZANDER

ZANDER works on partially observable probabilistic propositional planning domains consisting of a finite set of distinct *propositions*, any of which may be `true` or `false` at any discrete time t. A *state* is an assignment of truth values to these propositions. A possibly probabilistic *initial state* is specified by a set of decision trees, one for each proposition. *Goal states* are specified by a partial assignment to the set of propositions; any state that extends this partial assignment is a goal state. Each of a finite set 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 at time $t + 1$. A subset of the set of propositions is the set of *observable propositions*. The task is to find an action for each step t as a function of the value of observable propositions for steps before t and that maximizes the probability of reaching a goal state.

ZANDER translates the planning problem into an Ssat problem. Figure 2 shows an example of such an Ssat plan encoding (where all the unit clauses have been preprocessed). In this problem, a part must be painted, but the paint action succeeds only with probability 0.7 and it is an error to try to paint the part if it is already painted. The agent has two time steps, so the best plan is to paint the part at $t = 1$ and observe whether the action was successful, painting again (at $t = 2$) if it was not, and doing nothing (noop) otherwise.

The variables in an Ssat plan encoding fall into three segments [1]: the action-observation segment (variables pa_1, no_1, opd_1, pa_2, no_2 in Figure 2), the domain uncertainty segment (variables $cvp_1^{0.7}$, $cvp_2^{0.7}$ in Figure 2), and a segment representing the result of the actions taken given the domain uncertainty (variable pd_1 in Figure 2). The action-observation-history segment is an alternating sequence of existentially quantified variable blocks (one for each action choice) and randomly quantified variable blocks (one for each set of possible observations at a time step). We will refer to an instantiation of these variables as an *action-*

$$\exists pa_1 \exists no_1 \rotatebox[origin=c]{180}{R}\, opd_1 \exists pa_2 \exists no_2 \rotatebox[origin=c]{180}{R}\, cvp_1^{0.7} \rotatebox[origin=c]{180}{R}\, cvp_2^{0.7} \exists pd_1$$

$$\begin{array}{llll}
\{\ \{pa_1 \vee no_1\} & \wedge\ \{\overline{pa_1} \vee \overline{pd_1} \vee opd_1\} & \wedge\ \{pa_2 \vee no_2\} & \wedge \\
\{\overline{pa_1} \vee \overline{no_1}\} & \wedge\ \{\overline{pa_1} \vee pd_1 \vee \overline{opd_1}\} & \wedge\ \{\overline{pa_2} \vee \overline{no_2}\} & \wedge \\
\{\overline{cvp_1^{0.7}} \vee \overline{pa_1} \vee pd_1\} & \wedge\ \{\overline{pd_1} \vee pa_1\} & \wedge\ \{cvp_2^{0.7} \vee \overline{pa_2} \vee pd_1\} & \wedge \\
\{cvp_1^{0.7} \vee \overline{pa_1} \vee \overline{pd_1}\} & \wedge\ \{\overline{no_1} \vee \overline{opd_1}\} & \wedge\ \{\overline{pa_2} \vee \overline{pd_1}\} & \wedge \\
 & \wedge\ \{\overline{opd_1} \vee pa_1\} & \wedge\ \{pd_1 \vee pa_2\}\ \} &
\end{array}$$

Fig. 2. An example of an SSAT plan encoding, where pa_1 = (paint at $t = 1$), no_1 = (noop at $t = 1$), opd_1 = (observe painted after the action at $t = 1$), pa_2 = (paint at $t = 2$), no_2 = (noop at $t = 2$), $cvp_1^{0.7}$ = (chance variable associated with pa_1), $cvp_2^{0.7}$ = (chance variable associated with pa_2), and pd_1 = (painted at $t = 1$)

observation path. The domain uncertainty segment is a single block containing all the randomly quantified variables that modulate the impact of the actions on the observation and state variables. The result segment is a single block containing all the existentially quantified state variables. Essentially, ZANDER uses the solver described in Section 2 to find an assignment *tree* that specifies the assignments to existentially quantified action variables for all possible settings of the observation variables, such that the probability of satisfaction (which is also the probability that the plan will reach the goal) is maximized [1]. In what follows, we will refer to such a tree as an *action-observation tree*. We will also sometimes refer to existentially and randomly quantified variables as *choice* and *chance* variables, respectively.

4 APPSSAT

Before we describe APPSSAT it is worth looking at a previous approach to approximation in this framework. This approach illuminates some of the problems associated with formulating an approximation algorithm in this framework and explains some of the choices we made in developing APPSSAT. An algorithm called randevalssat that uses stochastic local search in a reduced plan space is described in [2]. The randevalssat algorithm uses random sampling to select a subset of possible chance variable instantiations (thus limiting the size of the contingent plans considered) and stochastic local search to find the best size-bounded plan. There are two problems with this approach. First, since chance variables are used to describe observations, a random sample of the chance 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*, and these inconsistencies can make it impossible to find a plan, even if one exists. Second, this algorithm returns a partial policy, that specifies actions only for those situations represented by paths in the random sampling of chance variables. APPSSAT addresses these two problems by:

1. designating each observation variable as a special type of variable, termed a *branch* variable, rather than a chance variable, and

2. evaluating the approximate plan's performance under all circumstances, not just those used to generate the plan.

The introduction of branch variables violates the pure SSAT form of the plan encoding, but is justified, we think, for the sake of conceptual clarity. We could achieve the same end in the pure SSAT form by making observation variables chance variables (as in [1]), and not including them when the possible chance-variable assignments are enumerated. But, rather than taking this circuitous route, we have chosen to acknowledge the special role played by observation variables; these variables indicate a potential branch in a contingent plan (hence the name). As such, the value of an observation variable node in the assignment tree described above is the *sum* of the values of its children. This introduces a minor modification into the ZANDER approach and has the benefit of clarifying the role of the observation variables.

APPSSAT incrementally constructs the optimal action-observation tree by updating the probabilities of the possible action-observation paths in that tree as it processes the instantiations of the chance variables. APPSSAT can stop this process after any number of chance variable assignments have been considered and extract and evaluate the best plan for the chance-variable assignments that have been considered so far (thus yielding an *anytime* algorithm). The current best plan is extracted by finding the tree of action-observation paths that has the highest probability and whose observations are consistent (note that this probability is a lower bound on the true probability of success of the plan represented by the tree). The probability of success of that plan is found by evaluating the full assignment tree using that plan.

If the probability of success of this plan is sufficient (probability 1.0 or exceeding a user-specified threshold), APPSSAT halts and return the plan and probability; otherwise, APPSSAT continues processing chance variable assignments. Note that the probability of success of the just-extracted plan can be used as a new lower threshold in subsequent plan evaluations, often allowing additional pruning to be done. The quality of the plan produced increases (if the optimal success probability has not already been attained) with the available computation time. See Figure 3 for a description of the algorithm.

Because the chance variable instantiations are investigated in descending order of probability, a plan with a relatively high percentage of the optimal success probability can potentially be found quickly. An exception is a domain in which the high probability situations are hopeless and the best that can be done is to construct a plan that addresses some number of lower probability situations. Even here, the basic SSAT heuristics used will allow APPSSAT to quickly discover that no plan is possible for the high-probability situations, and lead it to focus on the low-probability situations for which a plan is feasible. Of course, if *all* chance-variable assignments are considered, the plan extracted is the optimal plan, but, as we shall see, the optimal plan may sometimes be produced even after only a relatively small fraction of the chance-variable assignments have been considered.

```
APPSSAT ($\Phi$)
  nc = number of chance variables;
  k = pow(2, nc) = number of chance variable instantiations;
  d = number of chance variable instantiations processed per iteration;
  pc = current plan, initially empty;
  $\pi_{pc}$ = probability of success of the current plan, initially 0.0;
  $\pi_{thresh}$ = minimum acceptable probability of success;
  w = function that maps action-observation paths to probabilities,
      initially all 0.0;
  i = 0;
  while (i < k/d $\wedge \pi_{pc}$ < $\pi_{thresh}$);
    for j = (i * d) + 1 to (i * d) + d:
      cij = jth chance variable instantiation in descending order
            of probability;
      Pr[cij] = probability of chance variable instantiation cij;
      for each action-observation path (aop) that is consistent with cij:
        w(aop) = w(aop) + Pr[cij];
    pc = current best plan;
    $\pi_{pc}$ = Pr[pc reaches the goal];
  return pc and $\pi_{pc}$
```

Fig. 3. The APPSSAT algorithm for solving SSAT instances.

Unlike ZANDER, which, in effect, looks at chance variable instantiations at a particular time step based on the instantiation of variables (particularly action variables) at previous times steps, APPSSAT, by enumerating complete instantiations of the chance variables in descending order of probability, examines the most likely outcomes of all actions at all time steps. Because it is not taking variable independencies into account, it does so somewhat inefficiently. At the same time, however, by instantiating all the chance variables at the same time, APPSSAT reduces the SSAT problem to a much simpler SAT problem. Although this approach will also entail the repeated solving of a number of subproblems with one or more chance variable settings changed, the conjecture is that solving a large number of SAT problems will take less time than solving a large number if SSAT problems. Obviously, this will depend on the relative number of problems involved, but we have chosen to explore the approach embodied in APPSSAT first.

In the current implementation of APPSSAT, the user specifies the total number of chance-variable assignments to be considered, the interval at which the current plan should be extracted and evaluated (the default is 5% of the total number of chance-variable assignments being considered), and optional lower and upper success probability thresholds. If the algorithm finds a plan that meets or exceeds the upper success probability threshold, it halts and returns that plan.

All of the operations in APPSSAT can be performed as or more efficiently than the operations necessary in the ZANDER framework. The chance variable instantiations can be generated in descending order in time linear in the number of instantiations using a priority queue. Finding all consistent action-observation paths amounts to a depth-first search of the assignment tree checking for satisfiability using pruning heuristics (the central operation of ZANDER). Note also that once an action-observation path is instantiated, checking whether it can be extended to a satisfying assignment amounts to a series of fast unit propagations. In fact, once the chance variables have all been set, the remaining variables are all choice variables and the search for all action-observation paths that lead to satisfying assignments can be accomplished by any efficient SAT solver that finds all satisfying assignments. Extracting the current best plan involves a depth-first search of the action-observation tree, which is sped up by the fact that satisfiability does not have to be checked. Finally, plan evaluation requires a depth-first search of the entire assignment tree, but heuristics speed up the search, and the resulting probability of success can be used as a lower threshold if the search continues, thus potentially speeding up subsequent computation.

5 Results

Preliminary results are mixed but indicate that APPSSAT has some potential as an approximation technique. In some cases, it outperforms ZANDER, in spite of the burden of the additional approximation machinery. And, in those cases, where its performance is poorer, there is potential for further performance improvements (see Further Work).

We tested APPSSAT on three domains that ZANDER was tested on in [1]. The TIGER problem contains uncertain initial conditions and a noisy observation; the agent needs the entire observation history in order to act correctly. The COFFEE-ROBOT problem 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. Finally, the GO (GENERAL OPERATIONS) problem has no uncertainty in the initial conditions, but requires that probabilistic actions be interleaved with perfect observations. All experiments were conducted on an 866 MHz Dell Precision 620 with 256 Mbytes of RAM, running Linux 7.1.

In the 4-step TIGER problem, ZANDER found the optimal plan (0.93925 probability of success) in 0.01 CPU seconds. APPSSAT requires 0.42 CPU seconds to find the same plan (extracting and evaluating the current plan after every 5% of chance variable instantiations). This is, however, if we insist on forcing APPSSAT to look for the best possible plan (and, thus, to process all 512 chance variable instantiations), which seems somewhat out of keeping with the notion of APPSSAT as an approximation technique. If we run APPSSAT on this problem under similar assumptions, but specify a success probability threshold of 0.90 (we will accept any plan with a success probability of 0.90 or higher), APPSSAT returns a plan in 0.02 CPU seconds. The plan returned is, in fact, the optimal plan, and is found after examining the first 18 chance variable instantiations.

Table 1. Probability of success increases with number of chance variable instantiations

4-STEP TIGER			6-STEP COFFEE-ROBOT			7-STEP GO		
NCVI	SECS	PROB	NCVI	SECS	PROB	NCVI	SECS	PROB
1	0.0	0.307062	1	2.24	0.5	1	1.06	0.1250
2	0.0	0.614125	2	4.98	0.5	2	1.20	0.1250
3	0.0	0.614125	3	9.12	1.0	3	1.51	0.1250
4	0.0	0.668312	4	15.07	1.0	4	1.74	0.1250
5	0.01	0.668312	–	–	–	5	1.98	0.1250
6	0.01	0.722500	–	–	–	6	2.17	0.1250
7	0.01	0.722500	–	–	–	7	2.47	0.1250
8	0.01	0.722500	–	–	–	8	2.67	0.1250
9	0.01	0.776687	–	–	–	9	2.92	0.1250
10	0.01	0.776687	–	–	–	10	3.07	0.125
11	0.01	0.830875	–	–	–	11	3.36	0.1875
12	0.01	0.830875	–	–	–	12	3.62	0.1875
13	0.01	0.885062	–	–	–	13	3.83	0.1875
14	0.01	0.885062	–	–	–	14	4.03	0.1875
15	0.01	0.885062	–	–	–	15	4.26	0.1875
16	0.02	0.885062	–	–	–	16	4.47	0.1875
17	0.02	0.885062	–	–	–	17	4.83	0.1875
18	0.02	0.939250	–	–	–	18	4.97	0.1875
–	–	–	–	–	–	19	5.16	0.2500
–	–	–	–	–	–	20	5.44	0.2500

NCVI = number of chance variable instantiations
SECS = time in CPU seconds
PROB = probability of plan success

Table 1 provides an indication of what kind of approximation would be available if less time were available than what would be necessary to compute the optimal plan. This table shows how computation time and probability of plan success increases with the number of chance variable instantiations considered until the optimal plan is reached at 18 chance-variable instantiations.

The 6-step COFFEE-ROBOT problem provides an interesting counterpoint to the TIGER problem in that APPSSAT does better than ZANDER. ZANDER is able to find the optimal plan (success probability 1.0) in 19.34 CPU seconds, while APPSSAT can find the same plan in 9.12 CPU seconds. There are only 4 chance variable instantiations in the COFFEE-ROBOT problem and, since extraction and evaluation of the plan at intervals of 5% would result in intervals of less than one, the algorithm defaults to extracting and evaluating the plan after each chance variable instantiation is considered. Although one might conjecture that this constant plan extraction and evaluation is a waste of time, in this case it leads to the discovery of an optimal plan (success probability of 1.0) after processing the first 3 chance variable instantiations, and the resulting solution time of 9.12 CPU seconds (including plan extraction and evaluation time) is less than the solution time if we force APPSSAT to wait until all four chance variable instantiations have been considered before extracting and evaluating the best plan (15.07 CPU seconds).

This illustrates an interesting tradeoff. In the latter case, although APPSSAT does not extract and evaluate the plan after each chance variable instantiation, it does an extra chance variable instantiation, and this turns out to take more time than the extra plan extractions and evaluations. This is not surprising since checking a chance variable instantiation involves solving a SAT problem to find all possible satisfying assignments, while extracting and evaluating the plan requires only depth-first search. This suggests that we should be biased toward more frequent plan extraction and evaluation; more work is needed to determine if some optimal frequency can be automatically determined for a given problem. Table 1 provides an indication of how computation time and probability of plan success increases with the number of chance variable instantiations considered for the COFFEE-ROBOT problem. Interestingly, although the probability mass of the chance variables is spread uniformly across the four chance variable instantiations, APPSSAT is still able to find the optimal plan without considering all the chance variable instantiations.

The 7-step GO problem shows that this is not necessarily the case when, as in the GO problem, the probability mass is spread uniformly over many more (2^{21}) chance variable instantiations. In this problem, ZANDER is able to find the optimal plan (success probability 0.773437) in 2.48 CPU seconds. Because of the large number of chance variable instantiations to be processed, APPSSAT cannot approach this speed. APPSSAT needs about 566 CPU seconds to process 3000 (0.14%) of the total chance variable instantiations, yielding a plan with success probability of 0.648438. Table 1 provides an indication of how computation time and probability of plan success increases with the number of chance variable instantiations considered for the GO problem.

As the size of the problem increases, however, to the point where ZANDER might not be able to return an optimal plan in sufficient time, APPSSAT may be useful if it can return *any* plan with some probability of success in less time than it would take ZANDER to find the optimal plan. We tested this conjecture on the 10-step GO problem ($2^{30} = 1073741824$ chance variable instantiations). Here, ZANDER needed 405.35 CPU seconds to find the optimal plan (success probability 0.945313). APPSSAT was able to find a plan in somewhat less time (324.92 CPU seconds to process 20 chance variable instantiations), but this plan has a success probability of only 0.1875.

6 Further Work

We need to improve the efficiency of APPSSAT if it is to be a viable approximation technique, and there are a number of techniques we are in the process of implementing that should help us to achieve this goal. First, we are implementing an incremental approach: every time a new action-observation path is added, APPSSAT would incorporate that path into the current plan, checking to see if it changes that plan by checking values stored in that path from that point to the root. Whenever this process indicates that the plan has changed, the plan extraction and evaluation process will be initiated.

Second, when APPSSAT is processing the chance variable instantiations in descending order, in many cases the difference between two adjacent instantiations is small. We can probably take advantage of this to find the action-observation paths that satisfy the new chance variable instantiation more quickly.

Third, since we are repeatedly running a SAT solver to find action-observation paths that lead to satisfying assignments for the chance-variable assignments, and since two chance variable assignments will frequently generate the same satisfying action-observation path, it seems likely that we could speed up this process considerably by incorporating learning into APPSSAT. (We also note that we could improve performance by taking advantage of the speed available from current state-of-the-art SAT solvers.)

Finally, we are investigating whether plan simulation (instead of exact calculation of the plan success probability) would be a more efficient way of evaluating the current plan.

References

1. Majercik, S.M., Littman, M.L.: Contingent planning under uncertainty via stochastic satisfiability. Artificial Intelligence **147** (2003) 119–162
2. Littman, M.L., Majercik, S.M., Pitassi, T.: Stochastic Boolean satisfiability. Journal of Automated Reasoning **27** (2001) 251–296
3. Drummond, M., Bresina, J.: Anytime synthetic projection: Maximizing the probability of goal satisfaction. In: Proceedings of the Eighth National Conference on Artificial Intelligence, Morgan Kaufmann (1990) 138–144
4. Onder, N., Pollack, M.E.: Contingency selection in plan generation. In: Proceedings of the Fourth European Conference on Planning. (1997)
5. Boutilier, C., Dearden, R.: Approximating value trees in structured dynamic programming. In Saitta, L., ed.: Proceedings of the Thirteenth International Conference on Machine Learning. (1996)
6. Koller, D., Parr, R.: Computing factored value functions for policies in structured MDPs. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, The AAAI Press/The MIT Press (1999) 1332–1339
7. Koller, D., Parr, R.: Policy iteration for factored MDPs. In: Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI 2000). (2000)
8. Kearns, M.J., Mansour, Y., Ng, A.Y.: A sparse sampling algorithm for near-optimal planning in large markov decision processes. In: IJCAI. (1999) 1324–1331
9. Zhang, N.L., Lin, W.: A model approximation scheme for planning in partially observable stochastic domains. Journal of Artificial Intelligence Research **7** (1997) 199–230
10. Papadimitriou, C.H.: Games against nature. Journal of Computer Systems Science **31** (1985) 288–301