Faster Probabilistic Planning Through More Efficient Stochastic Satisfiability Problem Encodings

Stephen M. Majercik and **Andrew P. Rusczek**
Department of Computer Science

representation (each path through each decision tree is replaced by a *PPL* statement) but *PPL* allows the user to express planning problems in a more natural, flexible, and compact format. More importantly, *PPL* gives the user the opportunity (but does not require them) to easily express state invariants, equivalences, irreversible conditions, and action preconditions—information that can greatly decrease the time required to find a solution.

ZANDER converts the *PPL* representation of the problem into a *stochastic satisfiability* (SSAT) problem. An SSAT problem is a satisfiability (SAT) problem, assumed to be in conjunctive normal form, with two types of Boolean variables—termed *choice* variables and *chance* variables (Majercik 2000)—and an ordering specified for the variables. A choice variable is like a variable in a regular SAT problem; its truth value can be set by the planning agent. Each chance variable, on the other hand, has an independent probability associated with it that specifies the probability that that variable will be `True`.

Choice variables can be thought of as being existentially quantified—we must pick a single, best value for such a variable—while chance variables can be thought of as "randomly" quantified—they introduce uncontrollable random variation which, in general, makes it more difficult to find a satisfying assignment. So, for example, an SSAT formula with the ordering $v \exists w\ x \exists y\ z$ asks for values of v, x (as a function of w), and z (as a function of w and y) that maximize the probability of satisfaction given the independent probabilities associated with w and y. This dependence of choice variable values on the earlier chance variable values in the ordering allows ZANDER to map contingent planning problems to stochastic satisfiability. Essentially, ZANDER must find an assignment *tree* that specifies the optimal action choice-variable assignment given all possible settings of the observation variables (Majercik 2000).

The solver does a depth-first search of the tree of all possible truth assignments, constructing a solution subtree by calculating, for each variable node, the probability of a satisfying assignment given the partial assignment so far. For a choice variable, this is the maximum probability of its children. For a chance variable, the probability will be the probability weighted average of the success probabilities for that node's children. The solver finds the optimal plan by determining the subtree that yields the highest probability at the root node.

ZANDER uses unit propagation (assigning a variable in a unit clause—a clause with a single literal—its forced value) and, to a much lesser extent, pure variable assignment (assigning the appropriate truth value to a choice variable that appears only positively or only negatively) to prune subtrees in this search. Also, although the order in which variables are considered is constrained by the SSAT-imposed variable ordering, where there is block of similar (choice or chance) variables with no imposed ordering, ZANDER considers those with the earliest time index first. This *time-ordered heuristic* takes advantage of the temporal structure of the clauses induced by the planning problem to produce more unit clauses. ZANDER also uses dynamically calculated success probability thresholds to prune branches of the tree. We are currently working on incorporating learning to improve ZANDER's performance.

SSAT Encodings

The SSAT encoding currently used by ZANDER—a linear action encoding with classical frame axioms—and two of the alternate encodings described below—a linear action encoding with simple explanatory frame axioms and a parallel-action encoding—are similar to deterministic plan encodings described by Kautz, McAllester, & Selman (1996). A third encoding—a linear action encoding with complex explanatory frame axioms—contains elements of these two alternate encodings and arises due to the probabilistic actions in our domains.

In all cases, variables are created to record the status of actions and propositions in a T-step plan by taking three cross products: actions and times 1 through T, propositions and times 0 through T, and random propositions and times 1 through T. Let A, P, O, and R be the sets of actions, state propositions, observation propositions, and random propositions, respectively, and let $A = |A|$, $P = |P|$, $O = |O|$, and $R = |R|$. Let V be the set of variables in the CNF formula. Then:

$$|V| = (A + P + O + R)T + P \qquad (1)$$

The variables generated by all but the random propositions are choice variables. Those generated by the random propositions are chance variables. Each variable indicates the status of an action, proposition, observation, or random proposition at a particular time. In the parallel-action encoding, two additional actions are produced for each proposition $p \in P$ at each time: a maintain-p-positively action and a maintain-p-negatively action, which increases the number of variables in this encoding by $2P$.

Conceptually, we need clauses that enforce initial/goal conditions and clauses that model actions and their effects. The second group divides into two subgroups: clauses that enforce (or not) a linear action encoding, and clauses that model the impact of actions on propositions. Finally, in this last subgroup, we need clauses that model the effects of actions both when they *change* the value of a proposition and when they leave the value of a proposition *unchanged* (the frame problem). In the following sections, we will describe the clauses in each encoding that fulfill these functions.

Linear Action Encoding With Classical Frame Axioms

Initial and Goal Conditions: Let $IC^+ \subseteq P$ ($IC^- \subseteq P$) be the set of propositions that are `True`/`False` in the initial state, and $GC^+ \subseteq P$ ($GC^- \subseteq P$) be the set of propositions that are `True`/False in the goal state, where $IC^+ \cap IC^- = \emptyset$ and $GC^+ \cap GC^- = \emptyset$. This generates $O(P)$ unit clauses:

$$\bigwedge_{p \in IC^+} (p^0) \quad \bigwedge_{p \in IC^-} (\overline{p^0}) \quad \bigwedge_{p \in GC^+} (p^T) \quad \bigwedge_{p \in GC^-} (\overline{p^T}) \qquad (2)$$

where superscripts indicate times.

Mutual Exclusivity of Actions: Special clauses marked as "exactly-one-of" clauses specify that exactly one of the

literals in the clause be `True` and provide an efficient way of encoding mutual exclusivity of actions. A straightforward propositional encoding of mutual exclusivity of n actions would require, for each time t, an action disjunction clause stating that one of the actions must be `True`, and $\binom{n}{2} = O(n^2)$ clauses stating that for each possible pair of actions, one of the actions must be `False`. In subsequent solution efforts, the assignment of `True` to any action would force the assignment of `False` to all the other actions at that time step, but at a cost of discovering and propagating the effect of the $O(n)$ resulting unit clauses. Depending on the implementation of the SSAT solver, the number of mutual exclusivity clauses could also slow the discovery of unit clauses. By tagging the action disjunction clause as an exactly-one-of-clause, we reduce the total num-

If one considers the case in which every proposition and observation is unaffected by every action, then the upper-bound on the number of classical frame axioms generated in this encoding is $O(TA(O+P)) = O(TAP)$.

Linear Action Encoding With Simple Explanatory Frame Axioms

Clauses for initial and goal conditions, mutual exclusivity of actions, and action effects remain the same as for linear action encodings with classical frame axioms. But, since actions typically affect only a relatively small number of

Parallel Action Encoding

The final encoding seeks to increase the efficiency of the encoding by allowing parallel actions. Such encodings are attractive in that, by allowing for the possibility of executing actions in parallel, the length of a plan with the same probability of success can potentially be reduced, thereby reducing the solution time. For example, PGRAPHPLAN and TGRAPHPLAN (Blum & Langford 1999) allow parallel actions in probabilistic domains. PGRAPHPLAN and TGRAPHPLAN operate in the Markov decision process framework; although actions have probabilistic effects, the initial state and all subsequent states are completely known to the agent (unlike ZANDER, which can handle partially observable domains).

PGRAPHPLAN does forward dynamic programming using the planning graph as an aid in pruning search. ZANDER essentially does the same thing by following the action/observation variable ordering specified in the SSAT problem. When ZANDER instantiates an action, the resulting simplified formula implicitly describes the possible states that the agent could reach after this action has been executed. If the action is probabilistic, the resulting subformula (and the chance variables in that subformula) encode a probability distribution over the possible states that could result from taking that action. And the algorithm is called recursively to generate a new implicit probability distribution every time an action is instantiated.

TGRAPHPLAN is an anytime algorithm that finds an optimistic trajectory (the highest probability sequence of states and actions leading from the initial state to a goal state), and then recursively improves this initial trajectory by finding unexpected states encountered on this trajectory (through simulation) and addressing these by finding optimistic trajectories from these states to a goal state. In this sense, TGRAPHPLAN is more like APROPOS^2 (Majercik 2002), an anytime approximation algorithm based on ZANDER.

Clauses for initial and goal conditions remain the same. This encoding, however, does not include the action effects clauses or frame axioms common to the first three encodings. Instead, it works backward from the goal, in the manner of TGRAPHPLAN to generate *proposition-production* clauses enforcing the fact that propositions imply the disjunction of all possible actions that could produce that proposition, and clauses enforcing the fact that actions imply their preconditions. This process is complicated by the probabilistic nature of the domains, as we will explain in more detail below.

Note that this encoding requires the addition of a set of maintain actions, since one way of obtaining a proposition with a particular truth value is by maintaining the value of that proposition if it already has that value. Thus, the action set of this encoding (and the number of variables) is not strictly comparable to those of the previous encodings. These maintain actions imply, as a precondition, the proposition they maintain. These maintain-precondition clauses, along with the proposition-production clauses, implicitly provide the necessary frame axioms, since resolving a maintain-precondition clause with a proposition-production clause containing the maintain will produce a frame axiom. For example, the maintain-precondition clause:

$$\overline{\text{maintain-painted-positively}^t} \quad \textbf{painted}^{t-1} \tag{19}$$

and the proposition-production clause:

$$\overline{\textbf{painted}^t} \quad (\text{paint}^t \quad \text{cv}_\pi^t) \quad \text{maintain-painted-positively}^t \tag{20}$$

resolve to produce:

$$\overline{\textbf{painted}^t} \quad \textbf{painted}^{t-1} \quad (\text{paint}^t \quad \text{cv}_p^t) \tag{21}$$

which readily translates to the explanatory frame axiom:

$$\textbf{painted}^t \quad \overline{\textbf{painted}^{t-1}} \rightarrow (\text{paint}^t \quad \text{cv}_\pi^t) \tag{22}$$

Finally, now that we can take more than one action at a time we need clauses that specifically forbid the simultaneous execution of conflicting actions: Unlike TGRAPHPLAN, however, which makes two actions exclusive if *any* pair of outcomes interfere, the parallel action encoding we have developed for ZANDER makes two actions exclusive only under the circumstances in which they produce conflicting outcomes, as we will describe below.

Preconditions: Preconditions in the probabilistic setting are not as clear-cut as in the deterministic case. We will divide preconditions into two types. A *hard* precondition of an action is a precondition without which the action has no effect on any proposition under any circumstances. Let H_a^+ be the set of hard preconditions for action a that must be `True` and H_a^- the set of hard preconditions for action a that must be `False`. Then $O(TAP)$ clauses are generated:

$$\bigwedge_{t=1}^{T} \bigwedge_{a} \bigwedge_{A} \bigwedge_{h} \bigwedge_{H_a^+} (\overline{a}\, h^{t-1})$$
$$\bigwedge_{t=1}^{T} \bigwedge_{a} \bigwedge_{A} \bigwedge_{h} \bigwedge_{H_a^-} (\overline{a} \quad \overline{h^{1}} \tag{23}$$

Soft preconditions—action preconditions that are not hard but whose value affects the impact of that action—are modeled in the next set of clauses.

Proposition-Production Clauses: In these clauses, a proposition p implies the conjunction of all those actions (and the soft preconditions necessary for that action to produce p) that could have produced p.

This set of clauses is quite similar to the complex explanatory axioms, but, instead of providing explanations for *changes* in the status of a proposition, they provide explanations for the current status of the proposition. This leads to

$$\bigwedge_{|S|=1}^{} \bigwedge_{p} \bigwedge_{P_{-s+p-}} \bigwedge_{a} \bigwedge_{A_{+p-}}$$

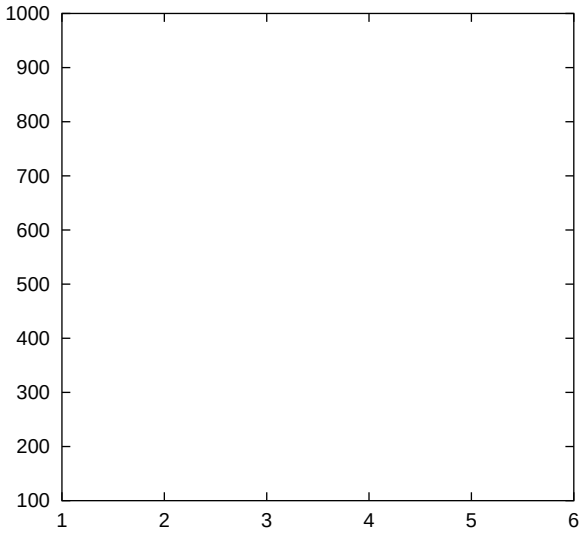
$$\text{cv}_{\pi}^{i} \quad \text{(24)}$$

and how propositions can become `False` (replace $\overline{p^{t-1}}$ with p^{t-1}). An example may help clarify the type of clauses produced and the use of the maintain action:

$$\begin{array}{rcl} \overline{p_8} & \rightarrow & (a_4 \quad p_3 \quad \overline{p_7} \quad \mathrm{cv}_5) \\ & & (a_5 \quad \overline{p_8} \quad p_{12} \quad p_{15} \quad \overline{\mathrm{cv}_{11}}) \\ & & (\text{maintain-}p_8\text{-negatively}) \end{array}$$

This set of clauses describes the possible ways that $\overline{p_8}$ can be produced. Of course, one of the actions that can produce a given proposition with a given truth value will always be the maintain action for that proposition and truth value.

Conflicting Actions: The absence of action effects clauses and the possibility of different outcomes of actions depending on the circumstances also means that clauses modeling action conflicts are somewhat more complex. Since actions can have very different effects on a proposition depending on the circumstances, actions can be conflicting under some sets of circumstances but not others and this must be incorporated into the action conflict clauses. For example, paint and maintain-error-negatively are conflicting if the object is already painted, but not otherwise,


1000
900
800
700
600
500
400
300
200
100
1
2
3
4
5
6

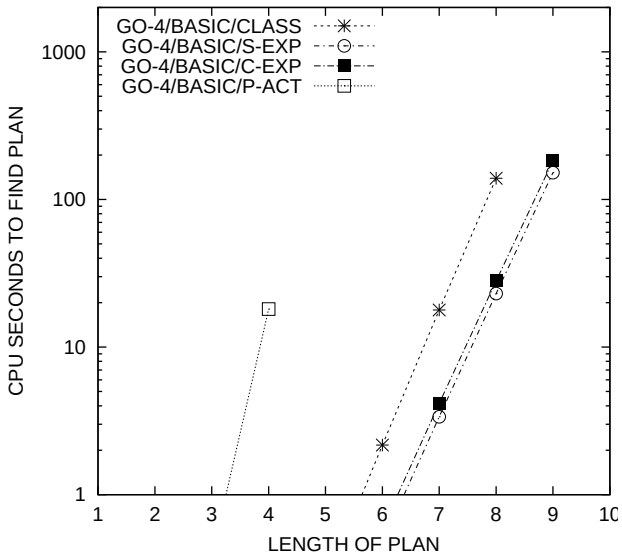


Figure 2: S-EXP and C-EXP encodings are the most efficient linear-action encodings; times shown here for the BASIC GO-4 domain.

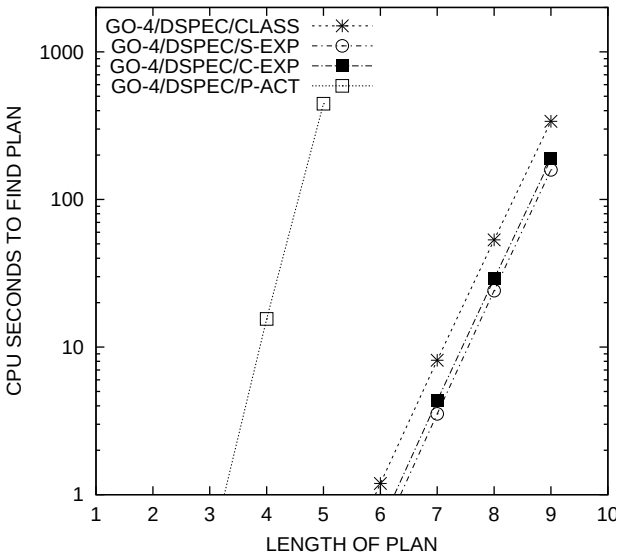


Figure 3: CLASS encodings become more competitive when domain-specific knowledge is added.

GO-4, and GO-5 domains, and this advantage increases with domain size. This is not surprising since, in any GO domain, all actions that have not had their intended effect can be done in parallel at any time step, and a larger number of actions translates directly into a greater benefit from parallelism. The increase in the number of variables and clauses—about 75% more variables and about 50-300% more clauses than the other encodings—is more than offset by the reduction in time steps necessary to achieve the same probability of success. Even in the GO-2 domain, the 5-step parallel plan produced by the P-ACT encoding succeeds with probability 0.938 and is found in 0.21 CPU second, compared to the 7-step linear-action plan produced by the linear-action encodings that succeeds with the same probability and is found in 0.11 to 0.26 CPU second. In the GO-5 domain, the 3-step and 4-step parallel-action plans produced by the P-ACT encoding succeed with a much higher probability (0.513 and 0.724 respectively) than the best plan produced by a linear-action encoding (0.363 for an 8-step plan), and the 3-step parallel-action plan solution time is an order of magnitude less than the best 8-step linear-action plan solution time (two orders of magnitude better than the 8-step CLASS encoding). The 2-step parallel-action plan succeeds with a probability of 0.237, which is slightly better than the success probability of the 7-step linear-action plan, and the 2-step parallel-action plan solution time is three orders of magnitude less than the 7-step linear-action plan solution times.

Adding Domain-Specific Knowledge

Not surprisingly, the addition of domain-specific knowledge (DSPEC encodings) significantly speeds up the solution process by making useful information explicit and, thus, more readily available to the solver. Kautz & Selman (1998) have explored the possibility of exploiting domain knowledge in deterministic-planning-as-satisfiability. In our test problems, we added knowledge of *irreversible conditions*: any fluent that is `True` (e.g. **painted**, **cleaned**, **polished**, or **error** in the GO-3 domain) at time t is necessarily `True` at time $t + 1$. This added knowledge is relatively minimal, adding one clause per fluent per time step. Yet, the addition of such clauses reduces the solution time by approximately 50-65% in some cases. The additional knowledge does nothing to improve the running time of the S-EXP or C-EXP encodings, since these encodings, by virtue of their explanatory frame axioms, already include clauses that model the persistence of positive propositions if there is no action that can negate them. In fact, the addition of these superfluous clauses frequently *increases* the running time of the S-EXP and C-EXP encodings. This is apparent in Figure 3 and Table 2(a), where, although the solution times for the S-EXP and C-EXP encodings are slightly worse, the CLASS encodings have become somewhat more competitive.

The benefit of adding domain-specific knowledge will certainly vary across domains and, in any case, the ease of adding such knowledge is critical. As mentioned earlier, various types of domain-specific knowledge can easily be added by the user in the form of *PPL* statements, but we are currently developing a domain analyzer that will automatically extract such information from the user's domain specification. In addition, the analyzer will also be able to extract temporal constraints implicit in the domain specification. Given the success of temporal-logic-based planners in recent AIPS planning competitions, we expect that the addition of such knowledge will improve performance considerably.

Further Work

Even more efficient SSAT encodings like the S-EXP encoding contain clauses that are superfluous since they sometimes describe the effects of an action that cannot be taken at a particular time step (or will have no impact if executed). We are currently working on an approach that is analogous to the GRAPHPLAN (Blum & Langford 1999) approach of incrementally extending the depth of the planning graph in the search for a successful plan. We propose to build the SSAT encoding incrementally, attempting to find a satisfactory plan in t time steps (starting with $t = 1$) and, if unsuccessful, using the knowledge of what state we *could* be in after time t to guide the construction of the SSAT encoding for the next time step. This reachability analysis would not only prevent superfluous clauses from being generated, but would also make it unnecessary to pick a plan length for the encoding, and would give the planner an *anytime* capability, producing a plan that succeeds with *some* probability as soon as possible and increasing the plan's probability of success as time permits.

There are two other possibilities for alternate SSAT encodings that are more speculative. Most solution techniques for partially observable Markov decision processes derive their power from a value function—a mapping from states to values that measures how "good" it is for an agent to be in each possible state. Perhaps it would be possible to develop a value-based encoding for ZANDER. If such an encoding could be used to perform value approximation, it would be particularly useful in the effort to scale up to much larger domains. The second possibility borrows a concept from belief networks to address the difficulty faced by an agent who must decide which of a battery of possible observations is actually relevant to the current situation. *D-separation* (Cowell 1999) is a graph-theoretic criterion for reading independence statements from a belief net. Perhaps there is some way to encode the notion of d-separation in an SSAT plan encoding in order to allow the planner to determine which observations are relevant under what circumstances.

References

Blum, A. L., and Langford, J. C. 1999. Probabilistic planning in the Graphplan framework. In *Proceedings of the Fifth European Conference on Planning*.

Cowell, R. 1999. Introduction to inference for Bayesian networks. In Jordan, M. I., ed., *Learning in Graphical Models*. The MIT Press. 9–26.

Giunchiglia, E.; Kartha, G. N.; and Lifschitz, V. 1997. Representing action: Indeterminacy and ramifications. *Artificial Intelligence* 95(2):409–438.

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

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. 2001. Stochastic Boolean satisfiability. *Journal of Automated Reasoning* 27(3):251—296.

Majercik, S. M. 2000. *Planning Under Uncertainty via Stochastic Satisfiability*. Ph.D. Dissertation, Department of Computer Science, Duke University.

Majercik, S. M. 2002. APROPOS^2: Approximate probabilistic planning out of stochastic satisfiability.

Onder, N. 1998. Personal communication.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press.

Schubert, L. 1990. Monotonic solution of the frame problem in the situation calculus; an efficient method for worlds with fully specified actions. In Kyburg, H.; Loui, R.; and Carlson, G., eds., *Knowledge Representation and Defeasible Reasoning*. Dordrecht: Kluwer Academic Publishers. 23–67.