Faster Probabilistic Planning Through More Efficient Stochastic Satisfiability Problem Encodings*

Stephen M. Majercik and **Andrew P. Rusczek**
Department of Computer Science
Bowdoin College
Brunswick, ME 04011-8486
{smajerci,arusczek}@bowdoin.edu

Abstract

The propositional contingent planner ZANDER solves finite-horizon, partially observable, probabilistic planning problems at state-of-the-art-speeds by converting the planning problem to a stochastic satisfiability (SSAT) problem and solving that problem instead (Majercik 2000). ZANDER obtains these results using a relatively inefficient SSAT encoding of the problem (a linear action encoding with classical frame axioms). We describe and analyze three alternative SSAT encodings for probabilistic planning problems: a linear action encoding with simple explanatory frame axioms, a linear action encoding with complex explanatory frame axioms, and a parallel action encoding. Results on a suite of test problems indicate that linear action encodings with simple explanatory frame axioms and parallel action encodings show particular promise, improving ZANDER's efficiency by as much as three orders of magnitude.

Introduction

Majercik (2000) showed that a compactly represented artificial intelligence planning domain can be efficiently represented as a *stochastic satisfiability* problem (Littman, Majercik, & Pitassi 2001), a type of Boolean satisfiability problem in which some of the variables have probabilities attached to them. This led to the development of ZANDER, an implemented framework that extends the planning-as-satisfiability paradigm to support contingent planning under uncertainty: uncertain initial conditions, probabilistic action effects, and uncertain state estimation (Majercik 2000).

There are different ways of encoding a probabilistic planning problem as an SSAT problem, however, and it is not obvious which encoding is best for which problem. In this paper, we begin to address the issue of producing maximally efficient SSAT encodings for probabilistic planning problems. In the next section, we describe ZANDER. In the next two sections, we describe four types of SSAT encodings of planning problems, analyze their size and potential benefits and drawbacks, and describe and analyze results using these encodings on a suite of test problems. In the final section, we discuss further work.

* Appears in the *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling*, pages 163-172, AAAI Press, 2002.

ZANDER

In this section, we provide a brief overview of ZANDER. Details are available elsewhere (Majercik 2000). ZANDER works on partially observable probabilistic propositional planning domains consisting of a finite set P of distinct *state propositions* which may be `True` or `False` at any discrete time t. A set $P' \subseteq P$ of state propositions is declared to be the set of observable propositions, and the members of O, the set of *observation propositions*, are the members of P' tagged as observations of the corresponding state propositions. Thus, each observation proposition has, as its basis, a state proposition that represents the actual state of the thing being observed.

A *state* is an assignment of truth values to the state propositions. An *initial state* is specified by an assignment to the state propositions. A probabilistic initial state is specified by attaching probabilities to some or all of the variables representing these propositions at time $t = 0$. *Goal states* are specified by a partial assignment to the set of state propositions; any state that extends this partial assignment is a goal state. Each of a finite set A of *actions* probabilistically transforms a state at time t into a state at time $t+1$ and so induces a probability distribution over the set of all states. The task is to find an action for each time t as a function of the value of observation propositions at times $t' < t$ that maximizes the probability of reaching a goal state.

Previous versions of ZANDER used a propositional problem representation called the sequential-effects-tree representation (ST), which is a syntactic variant of two-time-slice Bayes nets (2TBNs) with conditional probability tables represented as trees (Majercik 2000). In the ST representation, each action a is represented by an ordered list of decision trees, the effect of a on each proposition represented as a separate decision tree. This ordering means that the tree for one proposition can refer to old *and* new values of previous propositions, thus allowing the effects of an action to be correlated. The leaves of a decision tree describe how the associated proposition changes as a function of the state and action, perhaps probabilistically.

We currently represent problems using the Probabilistic Planning Language ($\mathcal{PPL}$). $\mathcal{PPL}$ is a high-level action language that extends the action language $\mathcal{AR}$ (Giunchiglia, Kartha, & Lifschitz 1997) to support probabilistic domains. An ST representation can be easily translated into a $\mathcal{PPL}$

representation (each path through each decision tree is replaced by a $\mathcal{PPL}$ statement) but $\mathcal{PPL}$ allows the user to express planning problems in a more natural, flexible, and compact format. More importantly, $\mathcal{PPL}$ 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 $\mathcal{PPL}$ 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 $\exists v \text{Я} w \exists x \text{Я} y \exists z$ 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 $\mathcal{A} = |A|$, $\mathcal{P} = |P|$, $\mathcal{O} = |O|$, and $\mathcal{R} = |R|$. Let V be the set of variables in the CNF formula. Then:

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

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 $2\mathcal{P}$.

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

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

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 $\binom{n}{2} = \mathcal{O}(n^2)$ 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 $\mathcal{O}(n)$ 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 number of clauses, and the solver can make the appropriate truth assignments to all actions in the clause as soon as one of them is assigned `True`. This generates $\mathcal{O}(T)$ exactly-one-of clauses:

$$\bigwedge_{t=1}^{T}\left(\bigvee_{a\in A} a^t\right) \tag{3}$$

Effects of Actions: Describing the effects of actions on propositions generates one or two clauses for each $\mathcal{PPL}$ action effect statement. If the statement is deterministic (a probability of 0.0 or 1.0), the statement generates a single clause modeling the action's deterministic impact on the proposition given the circumstances described by the statement. For example, the following clause states that an error condition is created if a painted part is painted again:

$$\text{paint causes error withp } 1.0 \text{ if painted} \tag{4}$$

The time indices are implicit; if the paint action is executed at time t and **painted** is `True` at time $t-1$, then **error** will be `True` at time t. Each $\mathcal{PPL}$ statement of the type:

$$a \text{ causes } p \text{ withp } \pi \text{ if } c_1 \text{ and } c_2 \text{ and } \ldots \text{ and } c_m \tag{5}$$

where $a \in A$, $p \in P \cup O$, $c_i \in P, 1 \leq i \leq m$, and π is 0.0 or 1.0 generates $\mathcal{O}(T)$ clauses:

$$\bigwedge_{t=1}^{T}\left(\overline{a^t} \vee \bigvee_{q\in P_{a+p}} \overline{q^{t-1}} \vee \bigvee_{q\in P_{a-p}} q^{t-1} \vee \overline{p^t}\right) \tag{6}$$

if $\pi = 0.0$, where $P_{a+p} \subseteq P$ is the set of c_is that appear positively in (5) and $P_{a-p} \subseteq P$ is the set of c_is that appear negatively in that statement. If $\pi = 1.0$, the final literal in each clause is p^t rather than $\overline{p^t}$.

If statement (5) is probabilistic ($0.0 < \pi < 1.0$), the statement generates two clauses modeling the action's probabilistic impact on the proposition given the circumstances described by that statement. An example will clarify this process. Suppose we have the following action effect statement:

$$\text{paint causes o-painted withp } 0.4 \text{ if new painted} \tag{7}$$

Here, the modifier "new" changes the implicit time index of **painted** to be the same as the time index of the action; in other words, "new painted" refers to the value of **painted** *after* the paint action has been executed.

Since the probability in this action effect statement is strictly between 0.0 and 1.0, a chance variable $\mathbf{cv}_1$ associated with this probability will be generated along with two clauses, one describing the impact of the action if the chance variable is `True` and one describing its impact if the chance variable is `False`. For example, this statement would generate the following two implications for time $t = 1$, where time indices, $-t$, are added to the variables:

$$\text{paint-1} \wedge \textbf{painted-1} \wedge \mathbf{cv}_{0.4}\text{-1} \rightarrow \textbf{o-painted-1} \tag{8}$$

$$\text{paint-1} \wedge \textbf{painted-1} \wedge \overline{\mathbf{cv}_{0.4}\text{-1}} \rightarrow \overline{\textbf{o-painted-1}} \tag{9}$$

where $\mathbf{cv}_{0.4}$-1 is the chance variable associated with this action effect, and the subscript indicates its probability. Negating the antecedent and replacing the implication with a disjunction produces two clauses:

$$\overline{\text{paint-1}} \vee \overline{\textbf{painted-1}} \vee \overline{\mathbf{cv}_{0.4}\text{-1}} \vee \textbf{o-painted-1} \tag{10}$$

$$\overline{\text{paint-1}} \vee \overline{\textbf{painted-1}} \vee \mathbf{cv}_{0.4}\text{-1} \vee \overline{\textbf{o-painted-1}} \tag{11}$$

Thus, each $\mathcal{PPL}$ statement of the type:

$$a \text{ causes } p \text{ withp } \pi \text{ if } c_1 \text{ and } c_2 \text{ and } \ldots \text{ and } c_m \tag{12}$$

where $a \in A$, $p \in P \cup O$, $c_i \in P$, and $0.0 < \pi < 1.0$ generates $\mathcal{O}(T)$ clauses:

$$\bigwedge_{t=1}^{T}\left(\overline{a^t} \vee \bigvee_{q\in P_{a+p}} \overline{q^{t-1}} \vee \bigvee_{q\in P_{a-p}} q^{t-1} \vee \overline{\mathrm{cv}_\pi^t} \vee \overline{p^t}\right) \tag{13}$$

where $P_{a+p} \subseteq P$ is the set of c_is that appear positively in (12) and $P_{a-p} \subseteq P$ is the set of c_is that appear negated in that statement, and cv_π is the chance variable generated by (12). The clauses in (13) model the effect of the action when the chance variable is `True`. A similar set of clauses, but with cv_π^t instead of $\overline{\mathrm{cv}_\pi^t}$ model the effect when the chance variable is `False`.

The set of propositions that determine the effect of an action a, $\bigcup_{p\in P_a}(P_{a+p} \cup P_{a-p})$, where P_a is the set of propositions affected by a, determine the arity C of that action. These propositions in the action effects statements can be either positive or negative, which leads to 2^C possible combinations of C propositions. If every action affects every proposition and every observation, both positively and negatively, then the number of action effects clauses per time step is $\mathcal{O}(\mathcal{A}(\mathcal{O} + \mathcal{P})2^C)$, where C is the maximum arity among all the actions in the domain. Since the number of propositions is always greater than or equal to the number of observations, this bound becomes $\mathcal{O}(\mathcal{AP}2^C)$. With T time steps, an upper-bound on the number of actions effects clauses is $\mathcal{O}(T\mathcal{AP}2^C)$.

Classical Frame Axioms: The fact that, for example, action a_1 has no impact on proposition p_1 is modeled explicitly by generating two action effect clauses describing this lack of effect: one clause describing a_1's nonimpact when p_1 is positive and one describing a_1's nonimpact when p_1 is negative. For every proposition $p \in P \cup O$ that an action $a \in A$ has no effect on, $\mathcal{O}(T)$ frame axiom clauses are generated:

$$\bigwedge_{t=1}^{T}\left(\overline{a^t} \vee \overline{p^{t-1}} \vee p^t\right) \wedge \bigwedge_{t=1}^{T}\left(\overline{a^t} \vee p^{t-1} \vee \overline{p^t}\right) \tag{14}$$

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 $\mathcal{O}(T\mathcal{A}(\mathcal{O}+\mathcal{P})) = \mathcal{O}(T\mathcal{A}\mathcal{P})$.

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 propositions, thus generating a large number of classical frame axioms, we replace the classical frame axioms with *explanatory frame axioms*. Explanatory frame axioms generate fewer clauses by encoding possible *explanations* for changes in a proposition. For example, if the truth value of proposition p_8 changes from `True` to `False`, it must be because some action capable of inducing that change was executed; otherwise, the proposition remains unchanged:

$$p_8^{t-1} \wedge \overline{p_8}^t \rightarrow a_4^t \vee a_5^t \vee a_7^t \qquad (15)$$

where a_4, a_5, and a_7 are the only actions at time t that can cause the proposition p_8 to change from `True` to `False`. We call these "simple" explanatory frame axioms because they do not make distinctions among the *possible* effects of an action. Unlike deterministic, unconditioned actions, it may be that, under certain circumstances, a_5 leaves p_8 unchanged; its presence in the above list merely states that there is a set of circumstances under which a_5 *would* change p_8 to $\overline{p_8}$. Thus, our simple explanatory frame axioms are similar to the frame axioms proposed by Schubert (1990) for the situation calculus in deterministic worlds, and like his frame axioms, depend on the *explanation closure assumption*: that the actions specified in the domain specify all possible ways that propositions can change.

In general, let $A_{+p-} \subseteq A$ be the set of all actions that, under some set of circumstances change proposition p's value from `True` to `False` and $A_{+p-} \subseteq A$ be the similar action set for the converse change in p's value. Then the following clauses are generated:

$$\bigwedge_{t=1}^{T} \bigwedge_{p\in P\cup O} \left(\overline{p^{t-1}} \vee p^t \vee \bigvee_{a\in A_{+p-}} a \right) \wedge \\ \bigwedge_{t=1}^{T} \bigwedge_{p\in P\cup O} \left(p^{t-1} \vee \overline{p^t} \vee \bigvee_{a\in A_{-p+}} a \right) \qquad (16)$$

Explanatory frame axioms must be generated for every proposition at every time step, regardless of the number of actions. Similar to the case of classical frame axioms, clauses must be generated to specify changing state from positive to negative as well as negative to positive. Therefore a tight upper-bound on the number of explanatory frame axioms generated is $\mathcal{O}(T(\mathcal{P}+\mathcal{O})) = \mathcal{O}(T\mathcal{P})$, a significant improvement over the $\mathcal{O}(T\mathcal{A}\mathcal{P})$ bound on classical frame axioms.

Linear Action Encoding With Complex 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 or simple explanatory frame axioms. The set of frame axioms is, however, different. A simple explanatory frame axiom specifies that a set of actions *could*, but do not always, effect a particular change in the truth value of a proposition. Complex explanatory frame axioms seek to improve the encoding of frame axioms by adding additional information that specifies *under what circumstances* these actions effect this change. These clauses are of the form:

$$\begin{aligned} p_8^{t-1} \wedge \overline{p_8}^t \quad \rightarrow \quad & (a_4^t \wedge p_3^{t-1} \wedge \overline{p_7}^{t-1}) \vee \\ & (a_5^t \wedge \overline{p_8}^{t-1} \wedge p_{12}^{t-1} \wedge p_{15}^{t-1}) \vee \\ & (a_7^t \wedge p_6^{t-1}) \qquad (17) \end{aligned}$$

which states that if the truth value of p_8 changes from `True` to `False` between times $t-1$ and t, either 1) action a_4 was executed at time t and p_3 was `True` and p_7 was `False` at time $t-1$, OR action a_5 was executed at time t and p_8 was `False` and p_{12} and p_{15} were `True` at time $t-1$ OR action a_7 was executed at time t and p_6 was `True` at time $t-1$. These complex explanatory frame axioms are very similar to the frame axioms proposed by Reiter (1991), and, again, depend on the explanation closure assumption of Schubert (1990).

In general, for a given proposition p there will be a set of actions $A_{+p-} \subseteq A$ that can change p's truth value from `True` to `False`. For each of these actions $a \in A_{+p-}$ there will be a set S of $\mathcal{PPL}$ statements of the form shown in (12) describing the ways in which a can effect this change. Let these statements be numbered from 1 to $|S|$. Let $P_{+s_ia+p-} \subseteq P$ be the set of propositions in statement s_i that must be `True` for action a to change p from `True` to `False`, and let $P_{-s_ia+p-} \subseteq P$ be the set of propositions in statement s_i that must be `False` for action a to change p from `True` to `False`. Finally, let $\mathrm{cv}_\pi^i, 1 \leq i \leq |S|$ be the chance variable, if any, in statement s_i. Then complex explanatory frame axioms of the following form will be generated:

$$\bigwedge_{t=1}^{T} \bigwedge_{p\in P\cup O} \overline{p^{t-1}} \vee p^t \vee \\ \bigvee_{a\in A_{+p-}} \bigvee_{i=1}^{|S|} \left(a \wedge \bigwedge_{p\in P_{+s_ia+p-}} p \wedge \bigwedge_{p\in P_{-s_ia+p-}} \overline{p} \wedge \mathrm{cv}_\pi^i \right) \qquad (18)$$

A similar set will be generated for actions that can change p's truth value from `False` to `True`.

The disjunction in the first line of (18) must be distributed over the disjunction of conjunctions in the second line in order to produce clauses, thus producing $\mathcal{O}(\mathcal{A}C2^C)$ clauses for each proposition and time step (where C is the maximum action arity) and $\mathcal{O}(T\mathcal{A}\mathcal{P}C2^C)$ complex explanatory frame axioms overall. In practice, however, subsumption usually reduces the number of these clauses to a much more reasonable level.

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:

$$\overline{\textsf{maintain-painted-positively}^t} \vee \textbf{painted}^{t-1} \tag{19}$$

and the proposition-production clause:

$$\overline{\textbf{painted}^t} \vee (\textsf{paint}^t \wedge \textrm{cv}_\pi^t) \vee \textsf{maintain-painted-positively}^t \tag{20}$$

resolve to produce:

$$\overline{\textbf{painted}^t} \vee \textbf{painted}^{t-1} \vee (\textsf{paint}^t \wedge \textrm{cv}_p^t) \tag{21}$$

which readily translates to the explanatory frame axiom:

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

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 $\mathcal{O}(T\mathcal{AP})$ clauses are generated:

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

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 $\mathcal{O}(T\mathcal{AP}C2^C)$ clauses describing how propositions can become `True`:

$$\bigwedge_{t=1}^{T} \bigwedge_{p \in P \cup O} \overline{p^{t-1}} \vee \\ \bigvee_{a \in A_{+p-}} \bigvee_{i=1}^{|S|} \left(a \wedge \bigwedge_{p \in P_{+s_i a+p-}} p \wedge \bigwedge_{p \in P_{-s_i a+p-}} \overline{p} \wedge \textrm{cv}_\pi^i \right) \tag{24}$$

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:

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

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, so we cannot have paint and **painted** at the same time step as maintain-error-negatively; i.e. $\overline{\textsf{paint} \wedge \textbf{painted} \rightarrow \textsf{maintain} - \textsf{error} - \textsf{negatively}}$. It would be incorrect to specify that these two actions can *never* be taken together. For each pair of actions a_1 and a_2, where a_1 has an action description statement s_i and a_2 has an action description statement s_j such that the hard and soft preconditions do not conflict, but the effects of s_i and s_j conflict, a clause is generated prohibiting the simultaneous execution of a_1 and a_2 under circumstances that are a superset of the union of the preconditions described in s_i and s_j:

$$\overline{\begin{array}{l} a_1 \wedge \bigwedge_{p \in P_{+s_i a_1 + p-}} p \wedge \bigwedge_{p \in P_{-s_i a_1 + p-}} \overline{p} \wedge \mathrm{cv}_i \rightarrow \\ a_2 \wedge \bigwedge_{p \in P_{+s_j a_2 + p-}} p \wedge \bigwedge_{p \in P_{-s_j a_2 + p-}} \overline{p} \wedge \mathrm{cv}_i \end{array}} \tag{25}$$

A loose upper bound on the number of such clauses produced is $\mathcal{O}(T\mathcal{P}(\mathcal{A}C2^C)^2)$.

Summary of Encodings

We will use the following notation to refer to the four types of encodings described above:

- CLASS: linear action encoding with classical frame axioms
- S-EXP: linear action encoding with simple explanatory frame axioms
- C-EXP: linear action encoding with complex explanatory frame axioms
- P-ACT: parallel action encoding

We briefly summarize the asymptotic bounds on the number of clauses in each encoding:

GO-n	Type of Encoding	Number of Variables	Number of Clauses	Average Clause Length
GO-2	CLASS	51	165	3.15
	S-EXP	51	135	3.09
	C-EXP	51	174	3.14
	P-ACT	88	276	3.17
GO-3	CLASS	71	290	3.14
	S-EXP	71	197	3.14
	C-EXP	71	268	3.35
	P-ACT	124	438	3.50
GO-4	CLASS	91	449	3.12
	S-EXP	91	261	3.15
	C-EXP	91	384	3.68
	P-ACT	160	650	3.98
GO-5	CLASS	111	642	3.10
	S-EXP	111	327	3.16
	C-EXP	111	542	4.23
	P-ACT	196	952	4.67

Table 1: Size of 5-step SSAT encodings of GO domains.

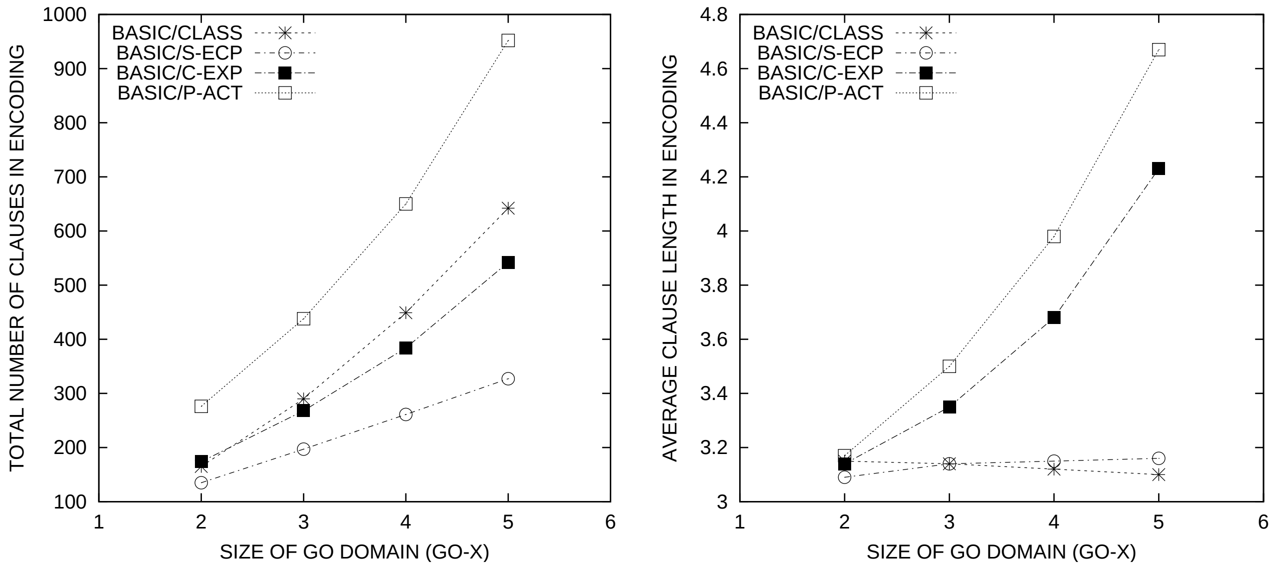
Type of SSAT Encoding	Total Clauses
CLASS	$\mathcal{O}(T\mathcal{A}\mathcal{P}2^C)$
S-EXP	$\mathcal{O}(T\mathcal{A}\mathcal{P}2^C)$
C-EXP	$\mathcal{O}(T\mathcal{A}\mathcal{P}C2^C)$
P-ACT	$\mathcal{O}(T\mathcal{P}(\mathcal{A}C2^C)^2$

In many cases, however, these bounds are fairly loose. To give a better picture of the relative sizes of encodings, we present encoding size information for 5-step plans for four versions of the GENERAL-OPERATIONS (GO) domain of increasing size in Table 1.

The S-EXP encoding appears to be the best encoding in many respects (Figure 1). It results in the fewest total clauses and grows at the slowest rate as the size of the GO domain increases (Figure 1(a)), and has an average clause size significantly shorter than the C-EXP and P-ACT encodings and almost as short as the CLASS encoding (Figure 1(b)). (Note that the decreasing average clause size of the CLASS encodings is due to the increasing number of short (3-literal) classical frame axioms and is not indicative of an advantage of the CLASS encodings.) Shorter clauses, of course, are more likely to produce more unit clauses faster, which speeds up the solution process. After the S-EXP encoding, the C-EXP, CLASS, and P-ACT encodings, in that order, produce larger encodings that grow at faster rates. Furthermore, while the average clause length for CLASS and S-EXP encodings remains roughly constant, the average clause length for C-EXP and P-ACT encodings grows at a much faster rate (Figure 1(b)).

Results and Analysis

The GO domain is a generic domain adapted from a problem described by Onder (1998). In this domain, there are an arbitrary number of actions, each of which produces a single desired effect with probability 0.5. The goal conditions require



(a) Total number of clauses (b) Average clause length

Figure 1: Clause statistics for the GO domains highlight the larger size of the P-ACT encoding.

that all these effects be accomplished without falling into an error condition, which results when the agent attempts to execute an action whose effect has already been achieved. For example, suppose we have three actions—paint, clean, and noop—the first two of which can produce a desired effect—**painted** and **cleaned**, respectively. There are three other propositions—**error**, **o-painted**, and **o-cleaned**—the last two of which are observations of the **painted** and **cleaned** propositions. In the initial state, **painted**, **cleaned**, and **error** are all `False`. The goal is to end up with the object painted and cleaned *exactly once*.

The GO domain has the advantage of scaling easily along several parameters: the number of actions, the number of propositions, the length of the plan. and the accuracy of the observations. We created versions of the GO domain that varied the number of actions from 2 to 5. From each of these four domains, we created two domains: a BASIC version that was essentially a translation of the corresponding ST representation, and a DSPEC version that added domain-specific knowledge. Each of these eight domains was translated into four SSAT encodings using the four encoding types described above: CLASS, S-EXP, C-EXP, and P-ACT. Finally, the plan length for each of these 32 types of encodings was varied from 1 to 10, producing 320 distinct encodings. (Although the GO domains are a very general type of probabilistic domain, we are currently running the same tests on a wider variety of domains to verify the generality of the results described below.)

Some idea of the relative size of the 32 encoding types modulo the plan length can be obtained from Table 1. Table 2(a) presents the running times of ZANDER for most of the 320 encodings on an 866 MHz Dell Precision 620 with 256 Mbytes of RAM, running Linux 7.1.[1] Table 2(b) presents the probability of success of the optimal plan for both the linear-action encodings and the parallel-action encoding at each plan length.

Analysis of Linear Action Encodings

The S-EXP encoding is clearly the best of the linear action encodings over the plan lengths tested (Table 2(a)). The average clause length of these encodings is slightly higher than the CLASS encodings (which always have the shortest average clause length) in the GO-4 and GO-5 domains, (Figure 1(b)), but this is offset by a significant reduction in the number of clauses (Figure 1(a)). An S-EXP encoding has 20-50% fewer clauses than a CLASS encoding (and the same number of variables), and this is reflected in the shorter run times for S-EXP encodings—typically 70-90% shorter than those for CLASS encodings (e.g. Figure 2 for the GO-4 domain).

The number of clauses in a C-EXP encoding falls in between that of CLASS and S-EXP encodings (Figure 1(a)), but the average clause size of a C-EXP encoding is larger than both of these encodings and this disparity grows with the size of the domain (Figure 1(b)), contributing to the relatively poor performance of the C-EXP encodings in larger domains. In addition, while the S-EXP encodings can be generated in less time than the CLASS encodings, C-EXP encodings take longer to generate due to the increased number and complexity of clauses.

Analysis of Parallel Action Encoding

The advantage of P-ACT encodings—the ability to take multiple actions at a time step and therefore construct and solve encodings for shorter length plans—is apparent in the GO-3,

[1]ZANDER's plan extraction mechanism is memory intensive; dashes in the table indicate that memory constraints prevented ZANDER from finding a solution.

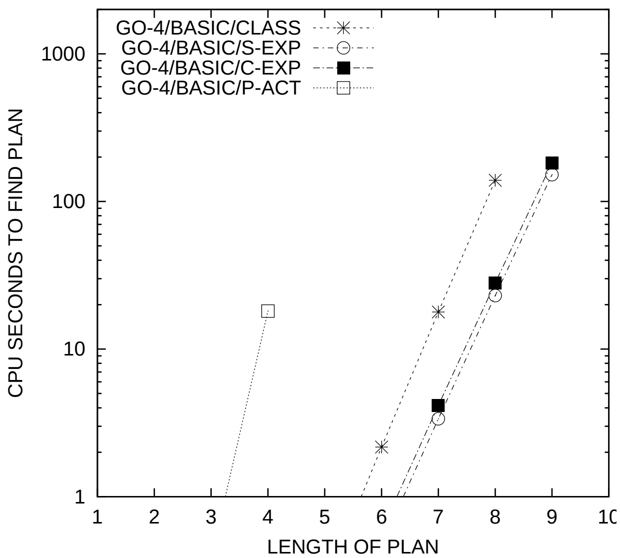


Figure 2: S-EXP and C-EXP encodings are the most efficient linear-action encodings; times shown here for the BASIC GO-4 domain.

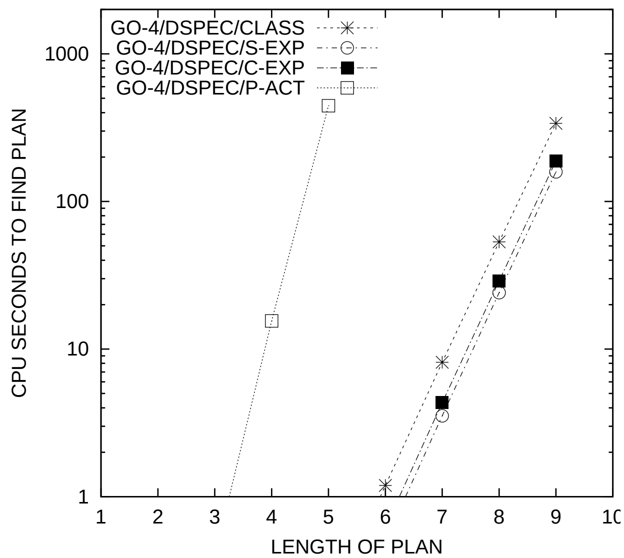


Figure 3: CLASS encodings become more competitive when domain-specific knowledge is added.

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 $\mathcal{PPL}$ 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.

GO-n	BASIC or DOMAIN SPECIFIC	Type of Encoding	Run Time in CPU Seconds by Plan Length (average of 5 runs)									
			1	2	3	4	5	6	7	8	9	10
GO-2	BASIC	CLASS	0.0	0.0	0.01	0.01	0.02	0.07	0.26	0.99	3.76	14.10
		S-EXP	0.0	0.0	0.0	0.01	0.01	0.03	0.11	0.39	1.31	4.44
		C-EXP	0.0	0.0	0.0	0.01	0.02	0.04	0.13	0.42	1.44	4.84
		P-ACT	0.0	0.0	0.01	0.03	0.21	1.38	8.76	52.88	304.19	–
	DSPEC	CLASS	0.0	0.0	0.0	0.0	0.01	0.04	0.14	0.47	1.61	5.41
		S-EXP	0.0	0.0	0.0	0.01	0.01	0.03	0.11	0.38	1.30	4.40
		C-EXP	0.0	0.0	0.0	0.01	0.01	0.04	0.13	0.43	1.46	4.89
		P-ACT	0.0	0.0	0.01	0.03	0.16	0.87	4.59	23.58	117.57	578.31
GO-3	BASIC	CLASS	0.0	0.0	0.0	0.01	0.08	0.47	2.81	16.17	90.04	490.03
		S-EXP	0.0	0.0	0.0	0.01	0.03	0.15	0.82	4.09	20.01	95.17
		C-EXP	0.0	0.0	0.0	0.01	0.04	0.19	0.95	4.70	22.60	107.37
		P-ACT	0.0	0.01	0.04	0.76	13.94	247.60	–	–	–	–
	DSPEC	CLASS	0.0	0.01	0.01	0.01	0.05	0.28	1.40	6.81	32.44	152.06
		S-EXP	0.0	0.0	0.01	0.01	0.04	0.16	0.85	4.24	20.54	98.45
		C-EXP	0.0	0.0	0.01	0.01	0.04	0.19	0.97	4.84	23.25	110.21
		P-ACT	0.0	0.0	0.04	0.68	9.01	105.83	–	–	–	–
GO-4	BASIC	CLASS	0.0	0.0	0.01	0.04	0.25	2.17	17.85	139.20	–	–
		S-EXP	0.0	0.0	0.0	0.01	0.06	0.47	3.37	23.05	151.88	–
		C-EXP	0.0	0.01	0.01	0.02	0.08	0.59	4.15	28.04	182.79	–
		P-ACT	0.0	0.01	0.38	18.11	850.37	–	–	–	–	–
	DSPEC	CLASS	0.0	0.01	0.01	0.02	0.17	1.19	8.14	53.23	338.21	–
		S-EXP	0.0	0.0	0.0	0.01	0.07	0.50	3.53	24.09	158.77	–
		C-EXP	0.0	0.0	0.01	0.01	0.08	0.62	4.33	29.00	188.41	–
		P-ACT	0.0	0.01	0.39	15.50	445.35	–	–	–	–	–
GO-5	BASIC	CLASS	0.0	0.0	0.01	0.07	0.74	7.93	81.58	810.90	–	–
		S-EXP	0.0	0.0	0.01	0.01	0.12	1.12	10.11	88.32	–	–
		C-EXP	0.0	0.01	0.01	0.02	0.18	1.56	13.77	117.98	–	–
		P-ACT	0.0	0.03	3.71	413.45	–	–	–	–	–	–
	DSPEC	CLASS	0.0	0.0	0.01	0.05	0.49	4.30	36.65	300.74	–	–
		S-EXP	0.0	0.0	0.01	0.02	0.13	1.19	10.69	92.83	–	–
		C-EXP	0.0	0.01	0.01	0.02	0.18	1.60	14.29	122.09	–	–
		P-ACT	0.0	0.03	3.74	349.61	–	–	–	–	–	–

(a) Execution times for the GO domains.

GO-n	Linear or Parallel Actions	Probability of Success of Optimal Plan by Plan Length									
		1	2	3	4	5	6	7	8	9	10
GO-2	LINEAR	0.0	0.250	0.500	0.688	0.813	0.891	0.938	0.965	0.981	0.989
	PARALLEL	0.250	0.563	0.766	0.879	0.938	0.969	0.984	0.992	0.996	0.998
GO-3	LINEAR	0.0	0.0	0.125	0.313	0.500	0.656	0.773	0.855	0.910	0.945
	PARALLEL	0.125	0.422	0.670	0.824	0.909	0.954	–	–	–	–
GO-4	LINEAR	0.0	0.0	0.0	0.063	0.188	0.344	0.500	0.637	0.746	–
	PARALLEL	0.063	0.316	0.586	0.772	0.881	–	–	–	–	–
GO-5	LINEAR	0.0	0.0	0.0	0.0	0.031	0.109	0.227	0.363	–	–
	PARALLEL	0.031	0.237	0.513	0.724	–	–	–	–	–	–

(b) Success probabilities for the GO plans produced.

Table 2: Test results for the GO domains.

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.