Nonchronological Backtracking in Stochastic Boolean Satisfiability*

Stephen M. Majercik
Bowdoin College
Brunswick ME 04011, USA
smajerci@bowdoin.edu

Abstract

Stochastic Boolean satisfiability (SSAT) is a generalization of satisfiability (SAT) that has shown promise as a vehicle for encoding and competitively solving probabilistic reasoning problems. We extend the theory and enhance the applicability of SSAT-based solution techniques by 1) establishing theoretical results that allow the incorporation of nonchronological backtracking (NCB), and lay the foundation for the incorporation of learning, into an SSAT solver, 2) implementing SOLVESSAT-NCB, an NCB-augmented SSAT solver, and 3) describing the results of tests of SOLVESSAT-NCB on randomly generated SSAT problems and SSAT encodings of probabilistic planning problems. Our experiments indicate that NCB has the potential to boost the performance of an SSAT solver, both in terms of time, yielding speedups of as much as five orders of magnitude, and space, allowing the solution of SSAT problems with larger solution trees. In some cases, however, NCB can degrade performance. We analyze the reasons for this behavior, present initial results incorporating a technique to mitigate this effect, and discuss other approaches to addressing this problem suggested by our empirical results.

1. Introduction

Stochastic Boolean satisfiability (SSAT), suggested by Papadimitriou [15] and explored further by Littman, Majercik & Pitassi [11], is a generalization of satisfiability (SAT) that is similar to quantified Boolean satisfiability (QSAT). 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.

Like QSAT, SSAT is PSPACE-complete, so it is theoretically possible to transform many probabilistic planning and reasoning problems of great practical interest into SSAT instances; e.g. plan evaluation, best polynomial-horizon plan, and Bayesian inferencing [11]. While such theoretically guaranteed translations are not always practical, previous work has shown that, in some cases, the transformation, and the solution of the resulting SSAT instance, can be done efficiently, providing an alternative approach to solving the original problem that is competitive with other techniques.

Majercik & Littman [12] showed that a probabilistic, partially observable, finite-horizon, contingent planning problem can be encoded as an SSAT instance such that the solution to the SSAT instance yields a contingent plan with the highest probability of reaching a goal state. They have used this approach to implement ZANDER, a probabilistic contingent planner competitive with three other leading approaches: techniques based on partially observable Markov decision processes, GRAPHPLAN-based approaches, and partial-order planning approaches [12]. Notably, ZANDER achieves this competitive level using a relatively naïve SSAT solver that does not attempt to adapt many of the advanced features that make today's state-of-the-art SAT solvers so efficient.

SSAT has also shown promise in the development of algorithms for *trust management* (TM) systems. TM systems are application-independent infrastructures that can enforce access control policies within and between organizations to protect sensitive resources from access by unauthorized agents. State-of-the-art TM systems have significant limitations, e.g. difficulty in expressing partial trust, and a lack of fault tolerance that can lead to interruptions of service. Freudenthal and Karamcheti have shown that stochastic satisfiability can form the basis of a TM system that addresses

* Appears in the *Proceedings of the Sixteenth International Conference on Tools With Artificial Intelligence*, pages 498-507, IEEE Press, 2004.

these limitations [5], and work is in progress to develop efficient solution techniques for the SSAT problems generated by such a system.

Roth [16] showed that the problem of belief net inferencing can be reduced to MAJSAT, a type of SSAT problem. Thus, in principle, an SSAT solver could solve the inferencing problem by solving the MAJSAT encoding of that problem. That this approach would be an *efficient* alternative to standard belief net inferencing algorithms is supported by recent work describing a DPLL-based approach to solving belief net inferencing problems that provides the same time and space performance guarantees as state-of-the-art exact algorithms and, in some cases, achieves an exponential speedup over existing algorithms [2, 1].

Thus, a more efficient SSAT solver would potentially give us better solution methods for a number of interesting, practical problems. In addition, the development of such a solver would be valuable for the insights it might provide into solving probabilistic reasoning problems, in general, and other PSPACE-complete problems, such as QSAT.

The SSAT solver that forms the basis of the planner ZANDER [12] is essentially the same as the solver described by Littman, Majercik & Pitassi [11]. This solver is an extension of the Davis-Putnam-Logemann-Loveland SAT solver [4], adapting unit propagation, pure variable elimination, and variable ordering heuristics to the SSAT framework. Lacking, however, are features, such as nonchronological backtracking, learning, and literal watching, that have been profitably incorporated into SAT [3, 14] and QSAT [8, 9, 10, 6] solvers. In this paper, we describe how nonchronological backtracking (NCB) can be incorporated into an SSAT solver. In Section 2, we describe the stochastic satisfiability problem. In Section 3, we present the theoretical results that allow the incorporation of NCB into an SSAT solver. In Section 4, we describe empirical results obtained by applying SOLVESSAT-NCB, an NCB-augmented SSAT solver, to randomly generated SSAT problems and SSAT encodings of probabilistic planning problems. In the last section, we discuss further work.

2. Stochastic Satisfiability

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* that is a Boolean formula constructed from these variables.[1] More specifically, the prefix $Q_1 v_1 \ldots Q_n v_n$ associates a quantifier Q_i, either existential ($\exists_i$) or randomized ($\mathbb{R}_i^{\pi_i}$), 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 this paper, we will use $x_1, x_2, \ldots$ for existentially quantified variables and $y_1, y_2, \ldots$ 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 $\overline{l}$ is the "opposite" of l, i.e. if l is v, $\overline{l}$ is $-v$; if l is $-v$, $\overline{l}$ 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:

$$\exists x_1, \mathbb{R}^{0.7} y_1, \exists x_2 \{\{x_1, y_1\}, \{x_1, \overline{y_1}\}, \{y_1, x_2\}, \{\overline{y_1}, \overline{x_2}\}\}.$$

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

1 NCB in SSAT is similar to NCB in QSAT, and, to make the similarities clear, we have adapted the terminology and notation used by Giunchiglia, Narizzano & Tacchella for NCB in QSAT [9].

$Pr[\Phi]$ formally. A partial assignment α of the variables V is a sequence of $k \leq n$ literals $l_1; l_2; \ldots; l_k$ 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. (As an aside, we note that any QSAT instance can be solved by transforming it into an SSAT instance—replace the universal quantifiers with randomized quantifiers—and checking whether $Pr[\Phi] = 1.0$.)

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. Then:

- If l is existential, setting $|l|$ such that l is `False` immediately produces an empty clause and $Pr[\Phi] = 0.0$, so the solver should always choose the forced value, and $Pr[\Phi] = Pr[\Phi(l)]$.
- If l is randomized, setting $|l|$ such that l is `False` immediately produces an empty clause and $Pr[\Phi] = 0.0$, so the solver should always choose the forced value, and the probability of satisfaction must be reduced by the probability of the forced value of the randomized variable; i.e. $Pr[\Phi] = Pr[\Phi(l)] \times Pr[l]$.

A literal l is *pure* if l is active and $\overline{l}$ is inactive. Then:

- If l is existential, setting the value of $|l|$ to *disagree* with l will falsify some literals in some clauses. Falsifying a literal in a clause makes it more likely that that clause will become empty, leading to a probability of satisfaction of 0.0, and so decreases (or leaves the same) the probability of satisfaction of the formula. The probability of satisfaction will be greatest when the value of $|l|$ is set to agree with l, regardless of whether the assignment is made now or when $|l|$ is encountered in the normal left-to-right evaluation of quantifiers (when l will still be pure), so $Pr[\Phi] = Pr[\Phi(l)]$.
- If l is randomized, there is no comparable simplification. Setting a pure randomized variable contrary to its sign in the formula can still yield a probability of satisfaction greater than 0.0, and this must be taken into account when computing the maximum probability of satisfaction.

For additional details, see [11].

Thresholding can also boost efficiency [11]. Thresholding computes *low* and *high* thresholds for the probability of satisfaction at each node in the solution tree. These thresholds delimit an interval outside of which it is not necessary to compute the exact value of the probability of satisfaction. Given a partial assignment α at a node, the solver can return "failure" if it establishes that $Pr[\Phi(\alpha)]$ is less than the low threshold, or "success" if it establishes that $Pr[\Phi(\alpha)]$ is greater than or equal to the high threshold. These thresholds sometimes allow the solver to avoid exploring the other value of a variable after it has explored the first value. In order to clarify the effect of NCB, we use only a very simplified version of thresholding in the SSAT solvers described in this paper: if the first branch of an existential variable returns a probability of satisfaction of 1.0, the solver can leave the other branch unexplored since the highest possible probability of satisfaction has already been attained.

These simplifications—unit and pure literals, and simplified thresholding—modify the rules given above for determining $Pr(\Phi)$, but we omit a restatement of the modified rules for the sake of brevity. These modified rules are

the basis for the following SSAT algorithm. For the sake of brevity, we omit the details of solution tree construction, indicating that only $Pr[\Phi]$ is returned. It is important to note, however, that the implementations of all SSAT algorithms in this paper construct and return the optimal solution tree as well, the size of which can be exponential in the number of randomized variables.

```
SOLVESSAT(Φ)
  if φ contains an empty clause: return 0.0;
  if φ is the empty set of clauses: return 1.0;
  if some l is an existential unit literal:
    return SOLVESSAT(Φ(l));
  if some l is a randomized unit literal:
    return SOLVESSAT(Φ(l)) × Pr[l];
  if some l is an existential pure literal:
    return SOLVESSAT(Φ(l));
  if the leftmost quantifier in Φ is existential and the
    variable thus quantified is v:
      if l is the literal corresponding to the first value
        of v explored and SOLVESSAT(Φ(l)) = 1.0:
        return 1.0;
      else:
        return max(SOLVESSAT(Φ(v)),
                   SOLVESSAT(Φ(−v)));
  if the leftmost quantifier in Φ is randomized and the
    variable thus quantified is v:
    return SOLVESSAT(Φ(v)) × Pr[v]+
           SOLVESSAT(Φ(−v)) × Pr[−v];
```

3. Nonchronological Backtracking

In the immediately preceding SSAT algorithm, it is almost always necessary to explore both branches of a variable v that is not unit or pure, the sole exception arising when v is existential and $Pr(\Phi(v)) = 1.0$. We will refer to the subtree resulting from the first value of v explored as the *left branch* of v, and the subtree resulting from the second value explored as the *right branch* of v. The goal of nonchronological backtracking (NCB) is to avoid exploring the right branch of a variable v whenever it can be determined that changing the value of v would not change the result obtained in the left branch. NCB does this by computing a *reason* whenever the current subproblem Φ' is determined to be unsatisfied or satisfied. Roughly, a reason is a collection of literals that is responsible for $Pr(\Phi')$. This reason is modified as the algorithm backtracks. At any point, the absence of a literal l in a reason indicates that the right branch of $|l|$ will return the same probability of satisfaction as the left branch, allowing the algorithm to avoid exploring the right branch. We will provide an example later in this section.

Given an SSAT problem Φ, we say that a partial assignment α satisfies Φ with probability p iff $Pr[\Phi(\alpha)] = p$. A reason $R \subseteq \{l | l \text{ is in } \alpha\}$ is a subset of the literals in a partial assignment α and is a reason for $Pr[\Phi(\alpha)] = p$ iff for every set of literals R' that agrees with α on R, $Pr[\Phi(R')] = p$. We say that reason R' *agrees* with α on R ($R' \sim_R \alpha$) if the following conditions are met:

1. Each literal $l' \in R'$ is either in α or is inactive in $\Phi(\alpha)$.
2. If $Pr[\Phi(\alpha)] < 1.0$, then every literal in R is also in R'.
3. If $Pr[\Phi(\alpha)] = 1.0$ then:
   (a) Every randomized literal in R is also in R'.
   (b) For every existential literal $l \in R'$, $\overline{l} \notin R$.

Suppose $R \subseteq \{l | l \text{ is in } \alpha\}$ is a reason for $Pr[\Phi(\alpha)] = p$. If $0 < p <= 1.0$, we will refer to R as a SAT reason; if $p = 0.0$, we will refer to R as an UNSAT reason. Using the terminology of Giunchiglia, Narizzano & Tacchella for NCB in QSAT [9], we will refer to backjumping with a SAT reason as *solution-directed* backjumping, and backjumping with an UNSAT reason as *conflict-directed* backjumping.

Given an SSAT problem Φ and a partial assignment α, a reason is created when $\phi(\alpha)$ contains an empty clause ($Pr[\Phi] = 0.0$) or when $\phi(\alpha)$ is the empty set of clauses ($Pr[\Phi] = 1.0$):

C1. If $\phi(\alpha)$ contains an empty clause C, $R = \{l | \overline{l} \in C'\}$ is a reason for $Pr[\Phi(\alpha)] = 0.0$, where C' is the clause in ϕ, the matrix of Φ, that has, at this point, become the empty clause C.

C2. If $\phi(\alpha)$ is the empty set of clauses, $R = \{l | l \in \alpha \wedge l \text{ is randomized}\}$ is a reason for $Pr[\Phi(\alpha)] = 1.0$. Note that R can sometimes be reduced further (thereby increasing the likely effectiveness of that reason) by eliminating literals from R which are not necessary for satisfiability.

Let us try to provide some intuition behind these rules. A reason is created at the end of a path in the solution tree; at this point the solver will begin to backtrack in order to calculate the impact of changing the value of any variable that appears in the partial assignment α leading to the current leaf, and whose second value has not yet been explored. Roughly speaking, we want to include a literal l from the current assignment α in the reason only if changing the value of the underlying variable $|l|$ could have an impact on the result obtained at that leaf.

The solver can choose the best value for an existential variable, so it needs to check the other value of such a variable only if that value might lead to a *higher* probability of satisfaction. For a randomized variable, the solver must take the weighted average of the two probabilities of satisfaction resulting from setting that variable to each possible value, so the solver needs to check the other value of the

variable both to see if that value will lead to a *higher* probability of satisfaction or if it will lead to a *lower* probability of satisfaction.

Thus, if the probability of satisfaction at a leaf is 1.0, the solver does not need to check the other value of existential variables in the assignment, but *does* need to check the other value of any randomized variable to see if it will lower the probability of satisfaction. Thus, the literals corresponding to these randomized variables should be in a SAT reason, subject to two important qualifications. First, the solver does not need to include *all* the randomized literals in the current assignment in the reason; any subset of the randomized literals that satisfies all those clauses not satisfied by some existential literal is sufficient. Second, existential literals may be drawn into the reason as the solver backtracks if they were indirectly responsible for the value of a randomized variable whose literal is in the reason.

If the probability of satisfaction at a leaf is 0.0, there will be a clause which has just become empty that is responsible for this. The solver needs to check the other value of any existential *or* randomized variable that was initially in this clause to see if the probability of satisfaction of 0.0 can be improved, either by making a better choice in the case of an existential variable, or by discovering that the other stochastically determined value of a randomized variable leads to a probability of satisfaction greater than 0.0. Thus, *all* the literals in the assignment that were responsible for the empty clause should be in an UNSAT reason. And, like the case for a SAT reason, other literals may be drawn into the reason while backtracking if they are found to be indirectly responsible for the variable values that led to the empty clause.

These intuitions underly the following rules for modifying reasons as the algorithm backtracks. Let Φ be an SSAT problem, l a literal, $\alpha; l$ an assignment for Φ, and R a reason for $Pr[\Phi(\alpha; l)] = p$.

M1. If l is an existential literal, then:

(a) if $l \notin R$, then R is a reason for $Pr[\Phi(\alpha)] = p$.

(b) if $l \in R$, then:

i. if $p = 1.0$, then $R \setminus \{l\}$ is a reason for $Pr[\Phi(\alpha)] = p$.

ii. if l is unit, then there will be a clause C such that $l \in C$ and for each $l' \in C$, if $l' \neq l$, $\overline{l'} \in \alpha$. Then $(R \cup \{l' | \overline{l'} \in C\} \setminus \{l, \overline{l}\})$ is a reason for $Pr[\Phi(\alpha)] = p$.

iii. if l is not unit, $\overline{l} \in R'$, and R' is a reason for $Pr[\Phi(\alpha; \overline{l})] = p'$, then $(R \cup R') \setminus \{l, \overline{l}\}$ is a reason for $Pr[\Phi(\alpha)] = \max(p, p')$.

M2. If l is a randomized literal, then:

(a) if $l \notin R$, then R is a reason for $Pr[\Phi(\alpha)] = p$.

(b) if $l \in R$, then:

i. if l is unit, then there will be a clause C such that $l \in C$ and for each $l' \in C$, if $l' \neq l$, $\overline{l'} \in \alpha$. Then $(R \cup \{l' | \overline{l'} \in C\} \setminus \{l, \overline{l}\})$ is a reason for $Pr[\Phi(\alpha)] = p \times Pr[l]$.

ii. if l is not unit, $\overline{l} \in R'$, and R' is a reason for $Pr[\Phi(\alpha; \overline{l})] = p'$, then $(R \cup R') \setminus \{l, \overline{l}\}$ is a reason for $Pr[\Phi(\alpha)] = p \times Pr[l] + p' \times Pr[\overline{l}]$.

iii. if l is not unit, $\overline{l} \notin R'$, and R' is a reason for $Pr[\Phi(\alpha; \overline{l})] = p'$, but $p = p'$, then R' is a reason for $Pr[\Phi(\alpha)] = p$.

Pure literals are not mentioned in these rules since they will never appear in a reason. A pure literal will never appear in a newly-created reason: SAT reasons are composed entirely of randomized literals, and a pure literal in an assignment is always existential. (As discussed earlier, there is no pure literal rule for randomized variables.) UNSAT reasons are composed of the `False` literals that have produced an empty clause; it could not be the case that one of these literals was pure when the assignment to its underlying variable was made, since a variable assignment induced by the pure literal rule never makes an active literal `False`. And, it is easy to show by induction that the rules for modifying reasons will never introduce a pure literal into a reason after it has been created.

The incorporation of these rules into the SOLVESSAT algorithm to produce the augmented SSAT solution algorithm SOLVESSAT-NCB closely follows the rules as stated and we omit a detailed description of the algorithm due to lack of space. The example in Figure 1, however, illustrates the operation of the NCB-augmented SSAT solver on the following SSAT problem:

$$\exists x_1, \mathbf{R}^{0.7} y_1, \exists x_2, \exists x_3 \quad \{\{\overline{x_1}, \overline{y_1}, x_2\}, \{x_1, y_1, \overline{x_2}, x_3\}, \{x_1, x_2, x_3\}, \{x_1, x_2, \overline{x_3}\}\}.$$

At the first (left-most) leaf, reached via the assignment $\overline{x_1}; \overline{y_1}; \overline{x_2}; x_3$, the formula is unsatisfied, i.e. ($Pr[\Phi(\overline{x_1}; \overline{y_1}; \overline{x_2}; x_3) = 0.0$) and SOLVESSAT-NCB creates an UNSAT reason composed of the literals in the assignment that made the clause $\{x_1, x_2, \overline{x_3}\}$ empty, i.e. $\{\overline{x_1}, \overline{x_2}, x_3\}$, (Rule **C1**) and backtracks to the x_3 node (i.e. the point at which a value is about to be assigned to x_3). The assignment $x_3 =$ `True` was forced since x_3 was unit, so the algorithm does not explore $x_3 =$ `False`, and we have $Pr[\Phi(\overline{x_1}; \overline{y_1}; \overline{x_2})] = 0.0$. The literals that forced this assignment (i.e. $\overline{x_1}; \overline{x_2}$ in the clause $\{x_1, x_2, x_3\}$) are substituted for x_3 in the reason, making the current reason $\{\overline{x_1}, \overline{x_2}\}$ (Rule **M1**(b)iii).

At the x_2 node, SOLVESSAT-NCB must explore the right branch of x_2 (i.e. $x_2 =$ `True`) since $x_2 =$ `False`) was *not* forced and x_2 *is* in the current reason. Setting x_2 to `True` forces x_3 to be set to `True`, and the formula is satisfied: $Pr[\Phi(\overline{x_1}; \overline{y_1}; x_2; x_3) = 1.0$. Here, SOLVESSAT-NCB

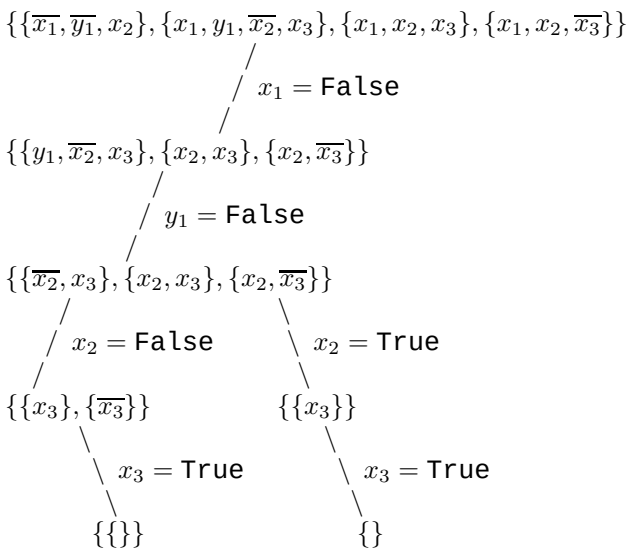


Figure 1. NCB can prevent useless exploration of subtrees in the solution tree

creates an empty reason $\{\}$, since existential variables are ignored in the creation of SAT reasons and $y_1 = \texttt{False}$ is not necessary for the satisfaction of any of the clauses (Rule **C2**). Backtracking to the x_3 node in this branch, $x_3 = \texttt{True}$ was forced since x_3 was unit, so the algorithm does not explore $x_3 = \texttt{False}$, and we have $Pr[\Phi(\overline{x_1}; \overline{y_1}; x_2) = 1.0$. (Note, however, that even if this were not the case, SOLVESSAT-NCB would not explore the other branch since x_3 is not in the current reason and the reason would remain the same, as indicated by Rule **M1**(a).)

When SOLVESSAT-NCB returns to the x_2 node, it picks the best value of x_2 (`True`) since x_2 is existential, and we have $Pr[\Phi(\overline{x_1}; \overline{y_1})] = 1.0$. The solver forms a new reason $\{\overline{x_1}\}$ by combining the $\{\overline{x_1}, \overline{x_2}\}$ reason from the $x_2 = \texttt{False}$ branch and the empty reason $\{\}$ from the $x_2 = \texttt{True}$ branch and deleting $\overline{x_2}$ (Rule **M1**(b)iv).

Normally, at the y_1 node, the solver would need to explore the $y_1 = \texttt{True}$ branch, but, since y_1 is not in the current reason SOLVESSAT-NCB does not explore this branch and the reason remains the same (Rule **M2**(a)). The absence of y_1 from the reason means that the probability of the $y_1 = \texttt{True}$ branch would be the same as that of the $y_1 = \texttt{False}$ branch ($Pr[\Phi(\overline{x_1}; y_1)] = Pr[\Phi(\overline{x_1}; \overline{y_1})] = 1.0$), so the probability of satisfaction is not reduced due to the randomized variable and we have $Pr[\Phi(\overline{x_1})] = 1.0$.

Backtracking to the x_1 node at the root, SOLVESSAT-NCB, according to the reason, would need to explore the $x_1 = \texttt{True}$ branch (since $\overline{x_1}$ is in the current reason). But, the fact that x_1 is an existential variable and $Pr[\Phi(\overline{x_1})] = 1.0$ allows the algorithm to avoid this additional work (simplified thresholding). For the sake of completeness, we note that Rule **M1**(b)i would remove literal $\overline{x_1}$ from the reason.

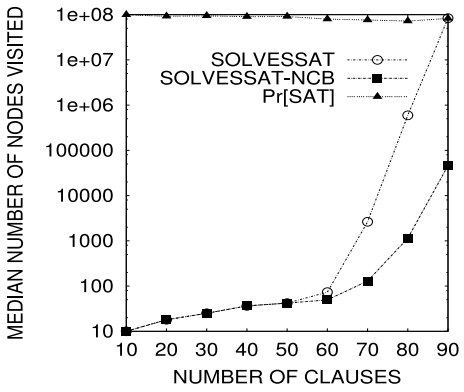
4. Experimental Results

We tested SOLVESSAT and SOLVESSAT-NCB on randomly generated SSAT problems and on SSAT encodings of probabilistic planning problems. All tests were run on a 1.8GHz Pentium 4 with 1GB RAM running Red Hat Linux 7.3.

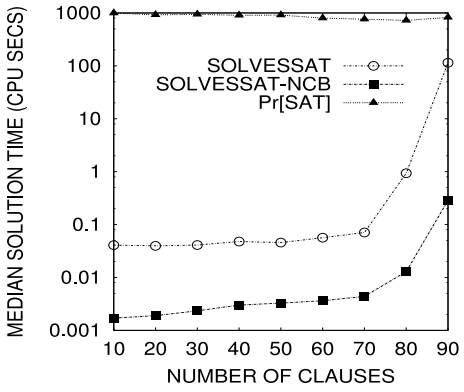
We generated random problems using a fixed-clause k-SSAT model similar to the k-QSAT Model A of Gent and Walsh [7]. We specify n, the number of variables, m, the number of clauses, l, the number of literals in each clause, k, the number of sequences of similarly quantified variables in the prefix, and b, the number of variables in each sequence of similar quantifiers. A clause is generated by randomly selecting a variable l times, discarding duplicates, and negating it with probability 0.5. Duplicate clauses are discarded.

Assembling a good set of randomly generated SSAT test problems is difficult. There are more parameters to vary in an SSAT problem than in a SAT problem; in addition to the number of variables, clauses, and literals per clause, one needs to generate a prefix. As noted in Section 2, changing this prefix can have a significant impact on the difficulty of the problem even when the rest of the problem remains the same. Our goal was to test SOLVESSAT-NCB on problems of increasing difficult by increasing the number of clauses, holding other parameters constant. In many cases, however, this produced a relatively sharp transition in difficulty, below which problems were "too easy" and the benefit of NCB was obscured by its overhead, and above which problems were "too hard" and in most cases could not be solved in a reasonable amount of time with or without NCB. We empirically generated a set of problems that avoided this phenomenon; the result of running SOLVESSAT and SOLVESSAT-NCB on these problems is shown in Figure 2.

NCB greatly improved the efficiency of the SOLVESSAT algorithm, decreasing the median number of nodes visited and the median solution time by approximately three orders of magnitude on larger problems (as high as five orders of magnitude in some problems). Just as important, SOLVESSAT-NCB was able to solve larger problems. Recall that all of the algorithms described in this paper return the solution tree as well as the optimal probability of satisfaction. As the number of clauses in the test problems increases, the average size of the solution tree increases as well. SOLVESSAT-NCB was able to solve a number of larger problems that SOLVESSAT could not solve due to insufficient memory (e.g. 39 out of the 100 100-clause prob-

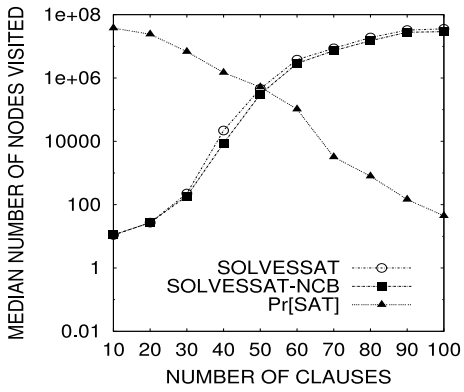


(a) Number of nodes visited

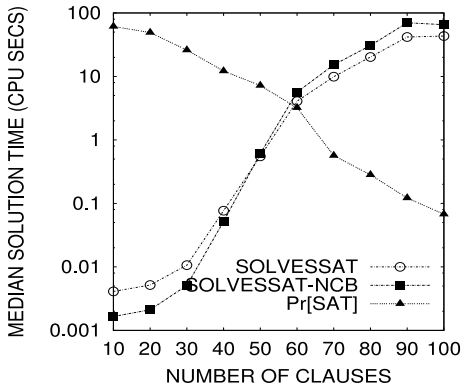


(b) Solution time in CPU secs

Figure 2. 10 blocks of 8 similarly quantified variables starting with $\exists$, 6 literals per clause, 100 problems at each clause level. $Pr[\Phi]$ scaled so top of graph is 1.0 and bottom is 0.0



(a) Number of nodes visited



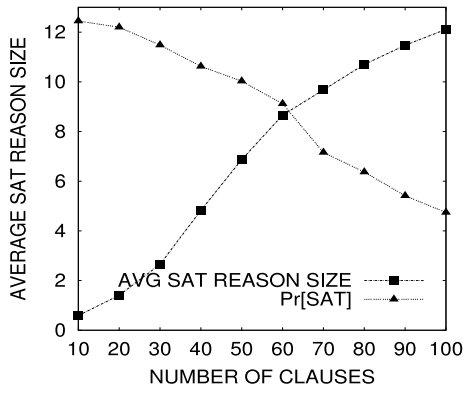
(b) Solution time in CPU secs

Figure 3. 10 blocks of 4 similarly quantified variables starting with $\exists$, 4 literals per clause, 100 problems at each clause level. $Pr[\Phi]$ scaled so top of graph is 1.0 and bottom is 0.0

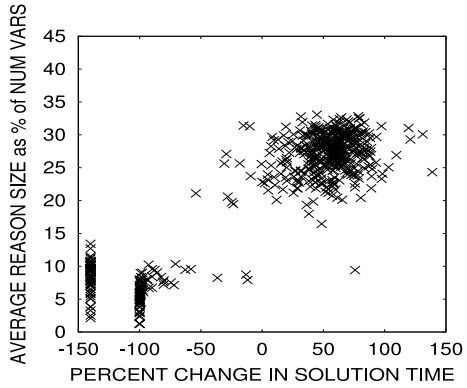
lems, which are not shown in the results since SOLVESSAT was unable to solve more than half of them).

In order to investigate the impact of NCB on the solution of problems in which the probability of satisfaction is farther away from 1.0, we attempted to create a set of problems that, as the number of clauses was increased, did not become too large too solve before the average probability of satisfaction began to decline. This turned out to be difficult for the size of problems in the initial test set so, to investigate the behavior of NCB as $Pr[\Phi]$ decreases, we generated a set of smaller problems in which the average $Pr[\Phi]$ decreases before the problems become too large to solve. The results (Figure 3) indicate that SOLVESSAT-NCB becomes less effective as $Pr[\Phi]$ decreases. Analysis indicated that this was due to an increase in SAT reason size (Figure 4(a)). Intuitively, when there are more clauses (constraints) it becomes more difficult to choose values for existential variables such that all paths in the tree of assignments result in satisfaction (and an overall $Pr[\Phi]$ of 1.0). An increasing number of clauses rely on randomized literals for satisfaction, producing a lower $Pr[\Phi]$ and increasing the number of randomized variables that must be included in a SAT reason. As the average SAT reason size increases, the effectiveness of these reasons decreases, both because more literals are included in the initial SAT reason, and because the larger initial reason makes it more likely that the algorithm will backtrack through a randomized unit literal l_u in the reason and need to add to the reason all the literals that forced the value of l_u (Rule **M2**(b)i).

Larger SAT reasons have a powerful effect because, even though a lower $Pr[\Phi]$ tends to reduce the number of SAT reasons, the percentage of SAT reasons never falls below approximately 90% and so dominates the overall average reason size. In fact, the overall average reason size is an ex-



(a)



(b)

Figure 4. (a) Average reason size for problems in Figure 3. $Pr[\Phi]$ scaled so top of graph is 1.0 and bottom is 0.0 (b) Average reason size as percentage of number of variables for all problems in Figs. 2 and 3

Problem	Horizon	SOL-SSAT	SOL-SSAT NCB	SOL-SSAT NCB-USR
G5	9	1.3e8	9.0e7	9.0e7
	10	9.2e8	6.5e8	6.5e8
CR	8	2.1e7	1.6e7	1.6e7
	9	1.4e8	1.0e8	1.0e8
TI	14	2.2e7	2.2e7	2.2e7
	15	6.4e7	6.4e7	6.4e7
SR	14	4.1e7	4.1e7	4.1e7
	15	1.2e8	1.2e8	1.2e8
M5	15	2.0e7	2.0e7	2.0e7
	16	4.3e7	4.3e7	4.3e7

(a) Number of nodes visited

Problem	Horizon	SOL-SSAT	SOL-SSAT NCB	SOL-SSAT NCB-USR
G5	9	176.3	174.5	147.6
	10	1193.2	1332.2	1069.9
CR	8	30.8	53.3	33.5
	9	215.1	364.9	229.2
TI	14	27.6	60.7	37.5
	15	82.2	186.1	114.0
SR	14	51.5	86.4	67.0
	15	153.0	263.7	202.2
M5	15	34.2	60.2	42.1
	16	73.4	120.2	91.3

(b) Solution time in CPU secs

G5 = GO-5
CR = COFFEE-ROBOT
TI = TIGER
SR = SHIP-REJECT
M5 = MEDICAL-5ILL

Figure 5. SOLVESSAT-NCB **does not improve performance significantly on a set of** SSAT **encodings of probabilistic contingent planning problems.**

cellent predictor of the effectiveness of NCB in these problems. Figure 4(b) plots the average reason size as a percentage of number of variables (ARSPV) against the percent change in solution time for the individual problems in Figs. 2 and 3. (The data points at -140% change in solution time are data points for problems that could not be solved by SOLVESSAT due to insufficient memory, but could be solved by SOLVESSAT-NCB.) If the ARSPV has a value of approximately 15% or less, then NCB will be beneficial; otherwise, the cost of NCB will outweigh its benefit. Furthermore, tests indicated that this statistic retains its predictive power even if calculated after only a few reasons have been created. We modified SOLVESSAT-NCB to calculate the ARSPV after 10 SAT reasons had been created and "turn off" NCB if the value of the ARSPV was greater than 15. In all but a handful of problems (out of 2000), the modified algorithm used NCB only when doing so would reduce the solution time, and the slight overhead of creating 10 reasons and checking the ARSPV value was negligible. We are conducting additional tests to determine if the value of this critical point is constant over a broad range of SSAT problems. Also, we conjecture that generating and using only UNSAT reasons when the critical ARSPV value is exceeded would be better than turning off NCB entirely, and we are currently testing this idea.

We also tested our algorithms on SSAT encodings of five probabilistic planning problems [12], varying the plan horizon. In most of the problems, NCB was able to reduce

the number of nodes visited only modestly (or not at all) (Figure 5(a)) and the overhead of NCB degraded performance (Figure 5(b)). Furthermore, the ARSPV value was not predictive of NCB effectiveness in these problems. The poor performance of NCB on these problems was surprising in light of the effectiveness of NCB in solving structured SAT problems [3] and structured QSAT problems [9] (although Giunchiglia, Narizzano & Tacchella [9] report some instances in which the overhead associated with NCB outweighs its benefits). Our analysis suggests three possible reasons for this poor performance, based on the characteristics of the SSAT encoding that result from the structure of the planning problem and the particular way the planning problem is translated into an SSAT instance. (There are multiple ways to encode planning problems as SSAT problems [13], and it is possible that another type of SSAT encoding would allow NCB to be more effective. More importantly, however, our analysis suggests that the characteristics of these SSAT problems that make them resistant to NCB are peculiar to planning problem encodings and not necessarily shared by all types of structured SSAT problems.)

First, the manner in which randomized variables are used in a plan encoding tends to undermine the effectiveness of solution-directed backjumping. A randomized variable y_i is usually associated with the probabilistic impact of a particular action on a particular fluent (proposition describing an aspect of the state) under certain conditions. This particular randomized variable typically appears in only two clauses modeling that action's effect under these conditions. If the action is taken and the conditions hold, the active literals in the two clauses look like: $\{\overline{y_i}, x_j\}, \{y_i, \overline{x_j}\}$, where x_j is the variable representing the fluent the action will affect. Either y_i and x_j will both be `True` or both be `False`. In either case, y_i will be necessary for any satisfying assignment that is an extension of this partial assignment. If the action is not taken, or if not all the conditions hold, both clauses will be satisfied and y_i becomes irrelevant (since it appears in only these two clauses) and is never even considered in the solution tree. In effect, it is removed in advance from any possible SAT reason. Since almost every randomized variable in a satisfying assignment is necessary for satisfaction and so must be included in the reason, the time spent trying to reduce the size of a SAT reason is wasted. We tested SOLVESSAT-NCB-USR, a version of SOLVESSAT-NCB in which all the randomized literals from a satisfying assignment are included in the SAT reason (the "unreduced" SAT reason). SOLVESSAT-NCB-USR was faster than SOLVESSAT-NCB, outperforming SOLVESSAT on the GO-5 problems, but the effect of NCB in the other problems remained disappointing (Figure 5).

The other reasons for the poor performance of NCB have to do with 1) the fact that most clauses encode a possible way that an action can change the status of a fluent from one time step to the next, and 2) the variable ordering. In these encodings, variables represent actions, observations, fluents, and the uncertainty associated with action outcomes. There is a set of these variables for each time step in the planning problem, tagged with a time index indicating what time step of the plan they are associated with. These variables or ordered such that the randomized variables that encode the uncertain outcomes of all actions in all time steps are in the next to inner-most sequence of similarly quantified variables in the ordering, and the existential variables representing all the fluents in all time steps are in the inner-most sequence of similarly quantified variables in the ordering. Within these last two blocks the variable ordering respects the time indices; variables with earlier time indices appear earlier in the ordering. These characteristics of the encoding have two consequences.

First, the values of almost all the fluent variables in the last block are set through the application of the unit literal rule. This is reflected empirically in a high ratio of unit literals to split variables (variables where both values are explored) in the solution process for these problems. This means that the values of almost all of these variables are forced; the other value will not be explored, even in the absence of NCB. Thus, NCB provides no benefit. Furthermore, all the literals that were responsible for making these literals unit will be added to the reason as the solver backtracks (Rules **M1**(b)(ii) and **M2**(b)(i)), lessening further the likelihood that large backjumps will occur. This is particularly relevant for UNSAT reasons, which, when first created, tend to be composed mostly of fluent literals.

Second, SAT reasons are composed of the randomized literals that are necessary for that particular satisfying assignment. But, since almost all the randomized variables are in a single block in the quantifier ordering, SAT reasons will tend to be composed of literals that are close to each other in the ordering, thus lessening the potential for large backjumps.

Note that some of the structure we have just described is due to the fact that these are encodings for partially observable planning problems. Even the GO-5 problem, which is completely observable, was encoded using the SSAT encoding framework for partially observable problems. Completely observable problems can be encoded with a variable ordering that intersperses the randomized variables encoding the uncertainty and the existential variables encoding fluents more evenly throughout the quantifier ordering. We are currently exploring whether NCB would be more effective on SSAT encodings of such problems.

5. Further Work

We described theoretical results that allow the incorporation of nonchronological backtracking into a stochastic satisfiability solver. Empirical results using an NCB-augmented SSAT solver based on this work indicate that the benefits of NCB are not as easy to realize in SSAT as QSAT. We described two responses to this problem in the preceding section; here we describe some alternatives currently under investigation that might allow us to realize the full potential of NCB.

A reduction in the overhead associated with NCB would make this technique more widely applicable, and we are exploring a more efficient way of creating and manipulating reasons using bit vectors.

A SAT reason can be any subset of those randomized literals that are necessary for the satisfaction of some clause. Choosing that subset with the aim of minimizing its size may not be the best strategy. Analysis of our empirical results indicates that, where there are choices, a criterion that favors randomized literals that appear in clauses with fewer existential literals may improve the efficiency of NCB, since such literals will have less of a tendency to pull existential literals into the reason during backtracking. The poor performance of NCB on SSAT encodings of planning problems, in which most clauses have at most one randomized literal, lends support to this notion.

In the absence of unit and pure literals, the algorithm considers both values of the next variable in the prefix. Since the order in which the algorithm considers variables in a block of similarly quantified variables does not affect the answer, problems with larger blocks of similarly quantified variables afford the opportunity to introduce selection heuristics that might improve the performance of NCB.

We suspect, however, that the full benefit of NCB cannot be realized in a solver that uses NCB alone. *Learning*, or recording the reasons as they are created in order to prevent the solver from rediscovering them in other branches of the computation, has been used to significantly enhance the performance benefits obtained from reason creation in both SAT [3] and QSAT [8] solvers and has the potential to greatly improve the effectiveness of NCB in an SSAT solver. UNSAT reasons can be recorded as additional clauses that prevent partial assignments leading to unsatisfiability; recording SAT reasons is not as straightforward. We are currently developing the infrastructure to support learning in SOLVESSAT-NCB.

Acknowledgments: We thank the anonymous reviewers for their helpful comments.

References

[1] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #sat and Bayesian inference. In *Proceedings of The 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS-2003)*, pages 340–351, 2003.

[2] F. Bacchus, S. Dalmao, and T. Pitassi. Value elimination: Bayesian inference via backtracking search. In *Proceedings of the Nineteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2003)*, pages 20–28, 2003.

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

[4] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[5] E. Freudenthal and V. Karamcheti. QTM: Trust management with quantified stochastic attributes. Technical Report TR2003-848, Courant Institute of Mathematical Sciences, New York University, New York, NY, 2003.

[6] I. Gent, E. Giunchiglia, M. Narizzano, A. Rowley, and A. Tacchella. Watched data structures for QBF solvers. In *Selected Papers from the Proceedings of the The Sixth International Conference on Theory and Applications of Satisfiability Testing (published in Lecture Notes in Computer Science #2919)*. Springer-Verlag, 2003.

[7] I. Gent and T. Walsh. Beyond NP: The QSAT phase transition. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 648–653. The AAAI Press/The MIT Press, 1999.

[8] E. Giunchiglia, M. Narizzano, and A. Tacchella. Learning for quantified Boolean logic satisfiability. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 649–654. The AAAI Press/The MIT Press, 2002.

[9] E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for quantified Boolean logic satisfiability. *Artificial Intelligence*, 145(1-2):99–120, 2003.

[10] R. Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In C. Fermueller and U. Egly, editors, *Proceedings of Tableaux 2002*, pages 160–175, 2002.

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

[12] S. M. Majercik and M. L. Littman. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, 147:119–162, 2003.

[13] S. M. Majercik and A. P. Rusczek. Faster probabilistic planning through more efficient stochastic satisfiability problem encodings. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling*, pages 163–172. AAAI Press, 2002.

[14] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Thirty-Ninth Design Automation Conference*, 2001.

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

[16] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1–2):273–302, 1996.