Initial Experiments in Stochastic Satisfiability

Michael L. Littman
Department of Computer Science
Duke University, Durham, NC 27708-0129
mlittman@cs.duke.edu

Abstract

This paper looks at the rich intersection between satisfiability problems and probabilistic models, opening the door for the use of satisfiability approaches in probabilistic domains. A generic stochastic satisfiability problem is examined, which can function for probabilistic domains as SAT does for deterministic domains. The paper defines a Davis-Putnam-Logemann-Loveland-style procedure for solving stochastic satisfiability problems, and reports on a preliminary empirical exploration of the complexity of the algorithm for a collection of randomly generated probabilistic problems. The results exhibit the familiar easy-hardest-hard pattern for the difficulty of random SAT formulae. Special cases of the stochastic satisfiability problem lie in different complexity classes, and one counterintuitive result is that the computational complexity and the empirical complexity of the problems examined do not track each other exactly—problems in the hardest complexity class are not the hardest to solve.

Introduction

There has been a recent focus in artificial intelligence (AI) on solving problems exhibiting various forms of uncertainty. In parallel, there is a great deal of work in AI and computer science on solving deterministic problems using techniques for testing Boolean satisfiability. Some recent work has looked at combinations of these ideas, viewing planning under uncertainty as stochastic Boolean satisfiability (Majercik & Littman 1998). This paper provides an approach for combining reasoning about uncertainty and satisfiability by exploring a framework that generalizes standard deterministic and stochastic satisfiability problems.

The remainder of this section reviews deterministic satisfiability and the following section introduces the stochastic satisfiability (SSAT) framework. The succeeding section describes the relationship between special cases of SSAT and planning and reasoning under uncertainty. The final sections describe a Davis-Putnam-Logemann-Loveland-based (DPLL) algorithm for solving SSAT problems and present empirical results applying this algorithm to randomly generated problems.

In deterministic satisfiability, or SAT, we are given a Boolean formula and wish to determine whether there is some assignment to the variables in the formula that results in the formula evaluating to "true." This problem is connected to problems throughout computer science from circuit design and complexity theory to AI. The last several years has seen tremendous progress in our ability to solve SAT problems, spurring interest in finding efficient ways to model problems such as planning (Kautz & Selman 1996) within the satisfiability framework.

Let $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle$ be a collection of n Boolean variables, and $\phi(\mathbf{x})$ be a k-CNF Boolean formula on these variables with m clauses. For example, $(x_1 + \overline{x_2} + x_4)(x_2 + x_3 + x_4)(\overline{x_1} + \overline{x_2} + \overline{x_3})$ is a $k = 3$-CNF formula with $n = 4$ variables and $m = 3$ clauses. This paper uses "1" and "0" for true and false, multiplication for conjunction, and addition for disjunction. Logical negation is written $\overline{x} = 1 - x$. With respect to a formula $\phi(\mathbf{x})$, an assignment to the Boolean variables $x_1, \ldots, x_n$ is *satisfying* if $\phi(\mathbf{x}) = 1$; a satisfying assignment makes the formula true. The decision problem SAT asks, given a Boolean formula $\phi(\mathbf{x})$ in 3-CNF, does it have a satisfying assignment? Or, symbolically, we want to know $\exists x_1, \ldots, \exists x_n (\phi(\mathbf{x}) = 1)$?

An interesting property of randomly generated formulae (Kirkpatrick & Selman 1994) is that when m is small relative to n, almost all formulae are satisfiable, and most algorithms find it easy to show this. When m is large relative to n, almost all formulae are unsatisfiable, and this is often relatively easy to show. For intermediate values of m (around $4.2n$), approximately half of the resulting formulae are satisfiable, and it is very difficult to show this. Thus, with respect to m, random SAT instances exhibit an "easy-hardest-hard" pattern. This pattern is consistent for various values of n, but becomes

more pronounced with larger n.

Stochastic Satisfiability

Papadimitriou (1985) explored an extension of SAT in which a randomized quantifier is introduced. The stochastic SAT (SSAT) problem is to evaluate a Boolean formula with both existential and randomized quantifiers:

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

In words, this formula asks whether there is a value for x_1 such that *for random* values of x_2 (choose 0 or 1 with equal probability), there exists a value of x_3, such that... the *expected* value of the Boolean formula $\phi(\mathbf{x})$ is at least a threshold θ. This type of satisfiability consists of alternating between making assignment choices for some variables and chance selection of assignments for other variables. The specification of an SSAT problem consists of the Boolean formula $\phi(\mathbf{x})$, the probability threshold θ, and the ordering of the quantifiers.

Note that the randomized quantifier R can be extended to allow arbitrary rational probability distributions over $\{0, 1\}$. The resulting generalization does not change any of the complexity results described in the next section, and is key to making a connection between SSAT, planning under uncertainty, and belief network inference.

Satisfiability and Uncertainty

As AI techniques are used more and more to attack real-world problems, many researchers have embraced probability theory as a way of representing the pervasive uncertainty they find. Two specific examples of this trend are the increasing use of Markov decision process models in planning and learning, and belief networks in reasoning and knowledge representation; however, the influence of probabilistic models in AI is felt quite broadly.

Whereas many basic deterministic problems are complete for the well-known complexity class NP, and are therefore formally equivalent to SAT, many planning and reasoning problems in probabilistic models lie in other complexity classes, such as #P or PP (Roth 1996; Littman, Goldsmith, & Mundhenk 1998). This means that, with respect to the current state of complexity theory, these problems cannot be reduced to SAT. However, many standard uncertain reasoning problems can be reduced to special cases of SSAT, which vary in complexity.

The NP-complete problem SAT is the SSAT problem obtained by using only existential quantifiers and setting $\theta = 1$. The problem of finding the most probable explanation (MPE) in a belief network (Dechter 1996) is equivalent to SAT. A related problem from planning under uncertainty is determining whether there is some choice of actions such that the most likely trajectory through state space to the goal exceeds a given probability threshold.

We define PP, or probabilistic polynomial time, by one of its complete problems, MAJSAT. In its standard formulation, MAJSAT asks, given a Boolean formula $\phi(\mathbf{x})$ in CNF, are at least half of its assignments satisfying? Thus, MAJSAT is concerned, not just with the existence, but the *number* of satisfying assignments.

This connects satisfiability to probability in the sense that, if we imagine that all assignments are equally likely, MAJSAT asks whether the probability of a satisfying assignment is at least 1/2. Thus, MAJSAT can be expressed as an instance of SSAT: $\text{R}x_1, \ldots, \text{R}x_n(E[\phi(\mathbf{x})] \geq 1/2)$ (or θ, more generally). Thus, MAJSAT is obtained from SSAT by using only randomized quantifiers.

This "decision" form of MAJSAT is polynomially equivalent to the problem of actually computing the probability of a satisfying assignment, since we can use binary search on θ to find the exact value of θ for which the probability is at least as big, but no bigger than θ. The class PP can be viewed as the decision problem version of #P, which actually *counts* the number of satisfying assignments.

The problem of belief network inference—given a belief network, values for its evidence nodes, and the value for a query node, what is the probability that the query node takes on the given value given the evidence?—is #P-complete (Roth 1996). Any belief network (with rational conditional probability tables) can be represented as a Boolean formulae. The reduction (Littman, Majercik, & Pitassi 1999) essentially consists of creating one variable per node in the belief network and one per conditional probability table entry. Clauses in the formula select a value for each belief network node depending on its parents' values. From this, it follows that the belief network inference decision problem ("Does the query node take on the given value with probability at least θ?") can be reduced to MAJSAT and is PP-complete. A PP-complete planning problem is plan evaluation in a probabilistic domain (Littman, Goldsmith, & Mundhenk 1998).

The complexity class NP^{PP} is formed by combining NP and PP. It is the class of problems that can be solved by guessing a solution (NP) and then performing a PP calculation for verification. A satisfiability problem that is complete for this class is E-MAJSAT ("exists" MAJSAT) (Littman, Goldsmith, & Mundhenk 1998), which combines elements of SAT and MAJSAT. An E-MAJSAT instance is defined by a CNF Boolean formula $\phi(\mathbf{x})$ on n Boolean variables, a threshold value θ, and a number $0 \leq c \leq n$. The decision problem is to report whether there is an assignment to the "choice" variables $x_1, \ldots, x_c$ so that the probability that the remaining chance variables $x_{c+1}, \ldots, x_n$ constitute

class	satisfiability problem Boolean formula	belief network problem planning problem
NP	SAT $\exists x_1, \ldots, \exists x_n (E[\phi(\mathbf{x})] \geq \theta)$	most probable explanation best trajectory
PP	MAJSAT $\forall x_1, \ldots, \forall x_n (E[\phi(\mathbf{x})] \geq \theta)$	belief updating (inference) plan evaluation
NP^{PP}	E-MAJSAT $\exists x_1, \ldots, \exists x_c, \forall x_{c+1}, \ldots, \forall x_n (E[\phi(\mathbf{x})] \geq \theta)$	maximum a posteriori hypothesis best polynomial size plan
PSPACE	SSAT $\exists x_1, \forall x_2, \ldots, \exists x_{n-1}, \forall x_n (E[\phi(\mathbf{x})] \geq \theta)$	influence diagrams best polynomial horizon plan

Table 1: Different arrangements of quantifiers result in SSAT problems complete for different complexity classes and correspond to basic problems in uncertain reasoning and planning.

a satisfying assignment of $\phi(\mathbf{x})$ is at least θ. Thus, E-MAJSAT is also an SSAT problem:

$$\exists x_1, \ldots, \exists x_c, \forall x_{c+1}, \ldots, \forall x_n (E[\phi(\mathbf{x})] \geq \theta).$$

Note that, if $c = n$, E-MAJSAT is simply a version of SAT, and, if $c = 0$, E-MAJSAT is exactly MAJSAT. In terms of complexity classes, NP $\subseteq$ PP $\subseteq$ NP^{PP}; E-MAJSAT is at least as hard as MAJSAT, which is at least as hard as SAT.

Other problems are NP^{PP}-complete, such as finding small satisfactory plans in uncertain domains (Littman, Goldsmith, & Mundhenk 1998) and generating "explanations" in belief networks. The belief network problems of calculating a maximum a posteriori (MAP) hypothesis or a maximum expected utility (MEU) solution (Dechter 1996) are also complete for NP^{PP}. In these problems, the choice variables correspond to the plan or explanation and the chance variables to the uncertainty.

The class PSPACE consists of the problems solvable using a polynomial amount of space. All the previously mentioned classes (NP, PP, NP^{PP}) can be solved in polynomial space by enumerating all assignments and combining the results in the appropriate way. The SSAT problem with alternating quantifiers and QBF are satisfiability problems that are PSPACE-complete. Note that each existential quantifier in an SSAT problem can be viewed as a type of maximization operator; the problem then becomes one of maximizing the probability that $\phi(\mathbf{x})$ is satisfied, given that some of the variables are under the control of "nature." This is equivalent to solving a finite-horizon partially observable Markov decision process (Papadimitriou & Tsitsiklis 1987). The problem remains PSPACE-complete when the domain is specified compactly via probabilistic STRIPS operators or an equivalent representation. Influence diagrams are a belief-network-like representation for the same problem.

Table 1 summarizes the relations between complexity classes and the stochastic satisfiability, belief network, and planning problems discussed.

An Algorithm for SSAT

The Davis-Putnam-Logemann-Loveland (DPLL) algorithm for Boolean satisfiability (Davis, Logemann, & Loveland 1962) works by enumerating partial assignments and monitoring for opportunities to simplify the formula. The use of pruning rules makes it possible to solve problems whose set of assignments could not be fully enumerated.

DPLL is designed to solve SAT problems, and, thus, only needs to deal with existential quantifiers. The algorithm described in this section can be viewed as an extension of the DPLL algorithm to SSAT by providing pruning rules for randomized quantifiers. Cadoli, Giovanardi, & Schaerf (1998) provide a set of pruning rules for universal quantifiers (for solving QBF problems); the pruning rules described below can be combined with theirs.

Define $\phi' = \text{simplify}(\phi, x_i, b)$, where ϕ' is the $(n-1)$-variable CNF formula obtained from assigning the single variable x_i the Boolean value b in the n-variable CNF formula ϕ and simplifying the result (including any necessary variable renumbering). Variables are numbered so that x_1 corresponds to the outermost quantifier and x_n to the innermost. Let $Q(x_i)$ be the quantifier associated with variable x_i.

An SSAT formula is defined by a set of numbered variables, a CNF formula, a threshold θ, and a mapping Q from variables to quantifiers. We define the *value* of an SSAT formula to be the value of the expression obtained by taking the SSAT expression and replacing $\exists$ with max and $\forall$ with average. This quantity is useful because it is greater than or equal to θ if and only the SSAT formula is true and less than or equal to θ if and only if it is false.

The evalssat algorithm in Figure 1 is a generalization of DPLL. It takes formula ϕ and low and high thresholds θ_l and θ_h (both initially set equal to θ). It returns a value less than θ_l if and only if the value of the SSAT formula is less than θ_l, a value greater than θ_h if and only if the value of the SSAT formula is greater than θ_h, and otherwise the exact value of the SSAT formula. Thus, it can be

```
evalssat(φ, θ_l, θ_h) := {
 if φ is the empty set, return 1
 if φ contains an empty clause, return 0
 /* Unit Resolution */
 if x_i is a unit variable with sign b and Q(x_i) = ∃,
  return evalssat(simplify(φ, x_i, b), θ_l, θ_h)
 if x_i is a unit variable with sign b and Q(x_i) = ∀,
  return evalssat(simplify(φ, x_i, b), 2θ_l, 2θ_h)/2
 /* Purification */
 if x_i is a pure variable with sign b and Q(x_i) = ∃,
  return evalssat(simplify(φ, x_i, b), θ_l, θ_h)
 /* Splitting */
 if Q(x_1) = ∃, {
  v_0 = evalssat(simplify(φ, x_1, 0), θ_l, θ_h)
  if v_0 ≥ θ_h, return v_0
  v_1 = evalssat(simplify(φ, x_1, 1), max(θ_l, v_0), θ_h)
  return max(v_0, v_1)
 }
 if Q(x_1) = ∀, {
  v_0 = evalssat(simplify(φ, x_1, 0), 2θ_l − 1, 2θ_h)
  if (v_0 + 1)/2 < θ_l, return v_0/2
  if v_0/2 ≥ θ_h, return v_0/2
  v_1 = evalssat(simplify(φ, x_1, 1), 2θ_l − v_0, 2θ_h − v_0)
  return (v_0 + v_1)/2
 }
}
```

Figure 1: The DPLL algorithm for satisfiability can be extended to solve SSAT problems.

used to solve the SSAT decision problem. Its basic structure is to compute the value of the SSAT formula from its definition; this takes place in the section labeled "*Splitting*", which enumerates all assignments, applying operators recursively from left to right. However, it is made more complex (and efficient) via pruning rules.

When a Boolean expression ϕ is evaluated that contains a variable x_i that appears alone in a clause in ϕ with *sign* b (0 if $\overline{x_i}$ is in the clause, 1 if x_i is in the clause), the normal right-to-left evaluation of quantifiers can be interrupted to deal with this variable. We call this case "*Unit Resolution*".

If the quantifier associated with x_i is existential, x_i can be eliminated from the formula by assigning it value b and recursing. As in DPLL, this is because assigning $x_i = 1 - b$ is guaranteed to make ϕ false, so $x_i = b$ can be no worse. If the quantifier associated with x_i is randomized, one branch of the computation will return a zero, so x_i can be eliminated from the formula by assigning it value b and recursing. The resulting value is divided by two, since it represents the value of only one branch.

The "*Purification*" pruning rule applies when there is a variable x_i that appears only with one sign b in ϕ. If $Q(x_i) = \exists$, the algorithm assigns $x_i = b$ and recurses. This is valid because any clause satisfied by an assignment with $x_i = 1 - b$ will also be satisfied by assigning $x_i = b$. Purification pruning does not appear possible for randomized variables as both assignments give *some* contribution to the value of the SSAT formula and must be considered independently.

Another useful class of pruning rules concerns the *threshold* parameters θ_l and θ_h. While some care must be taken to pass meaningful thresholds when applying unit resolution, threshold pruning mainly comes into play when variables are split to try to prevent recursively computing the value of both assignments to x_1. If $Q(x_1) = \exists$, after the first recursive call computing v_0, it is possible that θ_h has already been exceeded. In this case, the algorithm can simply return v_0 without ever computing v_1. In particular, it is possible that $v_1 > v_0$, but all that is significant is whether the larger of the two exceeds θ_h. If v_0 exceeds θ_l but falls short of θ_h, this can be used to increase the lower threshold for the recursive computation of v_1; since only the larger of v_0 and v_1 matters, the precise value of v_1 is not crucial if it less than v_0.

Threshold pruning is not as strong for randomized variables. There are two types of threshold pruning that apply. First, if assigning 0 to x_1 is sufficient to meet the threshold θ_h, then the algorithm need not recurse on assigning 1 to the variable: if $v_0/2 \geq \theta_h$, return $v_0/2$.

If the first value v_0 is so low that, even if $v_1 = 1$, $(v_0 + v_1)/2 < \theta_l$, then again v_1 need not be computed. If both tests fail, v_1 must be computed, but the thresholds can be adjusted accordingly.

This algorithm bears a close resemblance to searching AND/OR graphs, although this similarity has not yet been exploited. Enhancements based on variable-ordering heuristics are being explored.

Empirical Results

This section presents a set of preliminary experimental results on using the DPLL-based SSAT algorithm to solve random SSAT instances.

Throughout these experiments, the same SSAT algorithm is used. A set of 1,000 formulae with $n = 20$ variables and 141 clauses were randomly generated using `makewff` from AT&T Research. Thresholds were expressed as $\theta = 1/2^{n-t}$ for integer t in the range 0 to n; this defines t so that 2^t is the required number of satisfying assignments. Formulae with m clauses were created using the first m clauses from each of the 1,000 formulae.

The first experiment compares and contrasts MAJSAT with SAT. Figure 2 illustrates the average work required to solve random MAJSAT instances, varying the number of clauses and values of the threshold parameter θ.

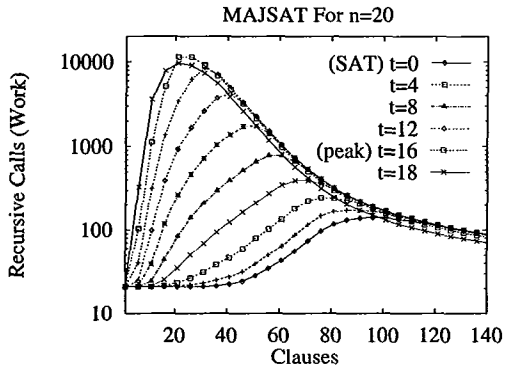

Figure 2: The difficulty of solving random MAJSAT instances varies with both the ratio of clauses to variables and with the satisfaction threshold (t).

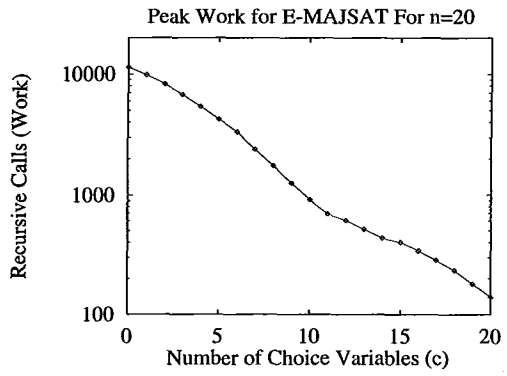

Figure 3: The difficulty of solving random E-MAJSAT instances varies with the number of choice variables c; more choice variables means easier problems.

The line on the plot labeled $t = 0$ is the curve for SAT; this problem is asking whether the probability of satisfaction is at least $1/2^n$, which is reached as long as there is even a single satisfying assignment. The classic easy-hardest-hard pattern is visible. In fact, nearly all the threshold values produce the same basic shape.

Note that the peak difficulty over all thresholds occurs at $t = 16$; it is much higher than the peak difficulty for SAT and occurs at a much lower setting of m. Instances with high values of the threshold are difficult because the threshold pruning rules for randomized quantifiers rarely apply—it is almost always necessary to check both branches to determine whether the probability threshold can be met.

That MAJSAT is more difficult to solve than SAT is not surprising, since it belongs to a higher complexity class. A more interesting pattern occurs in E-MAJSAT formulae. As mentioned earlier, the E-MAJSAT parameter c interpolates between MAJSAT ($c = 0$) and SAT ($c = n$). In terms of complexity theory, however, intermediate values of c place E-MAJSAT in a more difficult complexity class than either endpoint. This suggests analyzing the peak difficulty of E-MAJSAT as a function of c.

The following experiment was carried out. For each value of c from 0 to $n = 20$, the formulae were solved for each clause size m from 1 to 141 by 5s and threshold parameter t from 0 to 19. Work was averaged separately for each combination of settings, and the combination of values for t and m that resulted in the maximum average work was selected for each value of c. Figure 3 summarizes the results.

The results of these experiments are somewhat counterintuitive. Instead of the peak difficulty being obtained for $c = n/2$ as might be assumed from complexity theory, the difficulty is a *logarithmic* function of the number of choice variables. It

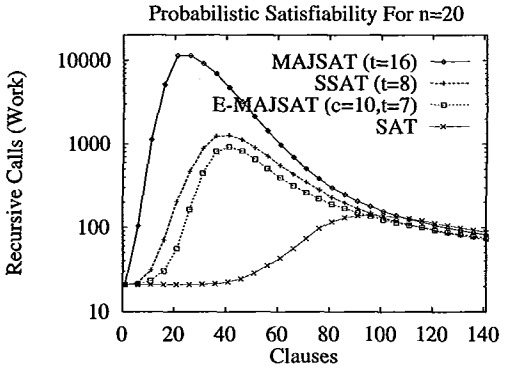

Figure 4: The peak difficulty of solving different SSAT instances varies with the structure of the operators and the number of clauses.

appears that the main effect is that each randomized quantifier changed to an existential quantifier results in a constant fraction of savings of work, perhaps due to the reduction in effective branching factor (see below).

Figure 4 plots the work versus number of clauses for each of four types of SSAT problems described earlier. For each problem class, the most difficult (on average) setting of the threshold parameter observed was used. There are several things to note here. One is that all these randomized satisfiability problems exhibit the same easy-hardest-hard pattern described for SAT. However, the number of clauses corresponding to the peak work differs for each problem. Also, the higher the peak, the smaller the number of clauses at the peak.

Another observation is that the relative peak difficulty of different problems does not match what might be predicted by complexity theory. In partic-

ular, as NP $\subseteq$ PP $\subseteq$ NP^{PP} $\subseteq$ PSPACE, we might expect SAT < MAJSAT < E-MAJSAT < SSAT in terms of peak work. In fact, the experiments come out with SAT < E-MAJSAT < SSAT < MAJSAT. That is, MAJSAT comes out as the hardest instead of the second easiest. This pattern can be observed with a range of values of n and k (Littman, Majercik, & Pitassi 1999).

The facts that E-MAJSAT is easier than MAJSAT and that SAT is easier than SSAT probably stem from the fact that randomized quantifiers are harder to prune than are existential quantifiers. SSAT and E-MAJSAT ($c = n/2$) both consist of half existential and half randomized quantifiers, and have very similar curves in Figure 4.

Conclusion and Future Work

This paper described a stochastic satisfiability framework, relating it to existing problems in reasoning and planning under uncertainty. It described a DPLL-based algorithm for solving satisfiability problems in this framework and showed that the algorithm exhibits interesting empirical behavior on randomly generated formulae.

Deterministic satisfiability problems can be solved in a number of different ways, including DPLL-based algorithms, resolution-based algorithms, and stochastic search. Dechter (1996) explores resolution-based solvers for an analogous set of problems to the ones described here. The practical utility of resolution-based methods for SSAT is not clear at present; unlike DPLL-style derivations, a resolution proof does not appear to yield an efficient procedure for value calculation. An important direction for research is combining random sampling (randomized quantifiers) and stochastic search (existential quantifiers) to solve SSAT problems (Littman, Majercik, & Pitassi 1999).

Another fruitful direction for studying SSAT problems is extending existing theoretical SAT results to the probabilistic setting. This would include probing critical behavior and scaling phenomena in random formulae (Kirkpatrick & Selman 1994) and proof-size-based lower bounds for exact algorithms (Beame & Pitassi 1996). The "stochastic satisfiability" approach is already producing probabilistic planners with state-of-the-art performance (Majercik & Littman 1999). This type of research promises insight into understanding what makes many uncertain reasoning problems hard to solve and identifying faster ways to solve them.

Acknowledgments. Thanks to Judy Goldsmith, Toni Pitassi, Pankaj Agarwal, Henry Kautz, Don Loveland, Julien Basch, John Reif, Bart Selman, Moises Goldszmidt, Toby Walsh, Ian Gent, Steve Majercik, and the reviewers for feedback and suggestions.

References

Beame, P., and Pitassi, T. 1996. Simplified and improved resolution lower bounds. In *37th Annual Symposium on Foundations of Computer Science*, 274–282. IEEE.

Cadoli, M.; Giovanardi, A.; and Schaerf, M. 1998. An algorithm to evaluate quantified Boolean formulae. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 262–267. The AAAI Press/The MIT Press.

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

Dechter, R. 1996. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)*, 211–219. Morgan Kaufmann Publishers.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1194–1201. AAAI Press/The MIT Press.

Kirkpatrick, S., and Selman, B. 1994. Critical behavior in the satisfiability of random Boolean expressions. *Science* 264:1297–1301.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The computational complexity of probabilistic plan existence and evaluation. *Journal of Artificial Intelligence Research* 9:1–36.

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

Majercik, S. M., and Littman, M. L. 1998. MAXPLAN: A new approach to probabilistic planning. In Simmons, R.; Veloso, M.; and Smith, S., eds., *Proceedings of the Fourth International Conference on Artificial Intelligence Planning*, 86–93. AAAI Press.

Majercik, S. M., and Littman, M. L. 1999. Contingent planning under uncertainty via probabilistic satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence.*

Papadimitriou, C. H., and Tsitsiklis, J. N. 1987. The complexity of Markov decision processes. *Mathematics of Operations Research* 12(3):441–450.

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

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