Speculative Pruning for Boolean Satisfiability

Wheeler Ruml, Adam Ginsburg, Stuart Shieber

Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
ruml@eecs.harvard.edu

January 20, 1998

Abstract

Much recent work on boolean satisfiability has focussed on incomplete algorithms that sacrifice accuracy for improved running time. Statistical predictors of satisfiability do not return actual satisfying assignments, but at least two have been developed that run in linear time. Search algorithms allow increased accuracy with additional running time, and can return satisfying assignments. The efficient search algorithms that have been proposed are based on iteratively improving a random assignment, in effect searching a graph of degree equal to the number of variables. In this paper, we examine an incomplete algorithm based on searching a standard binary tree, in which statistical predictors are used to speculatively prune the tree in constant time. Experimental evaluation on hard random instances shows it to be the first practical incomplete algorithm based on tree search, surpassing even graph-based methods on smaller instances.

1 Introduction

Satisfiability, determining whether a boolean formula could possibly be made true, is a quintessential NP-complete problem (Cook, 1971; Garey and Johnson, 1991). It remains hard even when the formula must be presented as a conjunction of disjunctions of at most three literals. An example of such a formula might be the rules for pet compatibility: $(dog \vee cat \vee parakeet) \wedge$

$(\neg cat \vee \neg dog) \wedge (\neg cat \vee \neg parakeet)$. This restricted version, known as 3-satisfiability or 3-SAT, has the maximum simplicity—the problem can be solved in polynomial time when restricted to two-element clauses (Even, Itai, and Shamir, 1976). Boolean satisfiability is particularly interesting because it is a fundamental task in reasoning with propositional logic: one can detect an inconsistent conclusion by conjoining it to the theory and showing that the resulting formula is unsatisfiable. It is also relatively easy to convert certain other NP-complete problems, such as graph coloring, circuit diagnosis, and STRIPS planning, into satisfiability problems.

In response to the intractability of satisfiability, two types of approximation algorithms have been proposed that trade a certain probability of an incorrect answer for enormous improvements in running time. The predictive algorithms use only general features of a formula, such as the average number of clauses per variable, to gauge its satisfiability. Although some of these algorithms can be guaranteed to run in time linear in the size of the formula, a predictor will not return an actual satisfying assignment; such assignments are useful in many applications. The second approach is to iteratively modify an initial unsatisfying assignment, with the intent of discovering one which satisfies the formula. These algorithms organize their search space as a graph rather than as a traditional binary tree, with assignments connected if they differ on the value of only a single variable. Since these methods do not attempt to consider all possible assignments, they sometimes erroneously conclude unsatisfiability when, in fact, a satisfying assignment exists.

In this paper, we will examine a third type of incomplete algorithm. Rather than making local moves over a graph representation of the possible assignments, we will use backtracking to move through a traditional binary tree. As with the graph-based algorithms, we will improve running time by ignoring portions of the search space. As we will see, we can use variants of the fast predictive satisfiability algorithms to determine whether a node warrants expansion. This yields a practical incomplete tree search algorithm for boolean satisfiability.

2 Previous Algorithms

The traditional complete tree search algorithm for satisfiability was described by Davis, Logemann, and Loveland (1962), and is known as DLL.[1]

[1] They implemented a modified version of the algorithm suggested by Davis and Putnam (1960).

Each node represents a partial assignment and nodes are expanded by assigning an additional variable. Along any path from the root, the formula is simplified as clauses are eliminated by becoming satisfied and as occurrences of variables (known as literals) become false and can be removed. Contradictory assignments are signalled by empty clauses. Any clause with a single literal forces the value of that variable; by performing this check, known as unit propagation, after every assignment, the effective depth of the tree is greatly reduced.

DLL runs quickly on many problems, and it was not until 1992 that Mitchell et. al. formulated a distribution of instances, known as random 3-SAT, for which DLL requires exponential time in the average case. In this model, all clauses contain three literals of three distinct variables, each negated with probability $\frac{1}{2}$, and no clause is repeated. The number of clauses per variable, known as β, can be varied, producing easily satisfied instances with few constraints, overconstrained instances with easily found contradictions, or difficult problems, with about $4.258n + 58.26n^{-2/3}$ clauses per variable (Crawford and Auton, 1996; Mitchell, Selman, and Levesque, 1992). Even on difficult problems, however, careful selection of the next variable to assign can lower the exponent: Crawford and Auton's TABLEAU algorithm runs in $O(2^{n/19.5})$ time in the average case on hard random 3-SAT instances (Crawford and Auton, 1996).

Running DLL may not be necessary if one is willing to tolerate some probability of error. The WSAT algorithm of Selman, Kautz, and Cohen (1994) uses a very different kind of search from DLL. Instead of organizing partial assignments into a tree and thereby implicitly recording the portion of the possibilities that have been considered, WSAT modifies a complete assignment without attempting to record its efforts. In effect, it makes local moves through a graph of assignments. The algorithm starts with a random assignment and repeatedly selects an unsatisfied clause and flips the assignment of one of its variables. If one of the variables can be changed without causing another clause to become unsatisfied, then that variable is flipped, otherwise, with probability $1 - p$, the variable that causes the fewest other clauses to become unsatisfied is flipped, and with probability p, a random variable in the clause in flipped. If a satisfying assignment is not found after some number of flips, the algorithm tries again with a new random assignment. WSAT runs in $O(n^{-0.6+0.4\ln(n)})$ time in the average case on hard random 3-SAT instances that are known beforehand to be satisfiable (Parkes and Walser, 1996).

If one is willing to tolerate error and also doesn't require an actual satisfying assignment, even simpler predictive algorithms are available. As

explained above, when given a formula from the random 3-SAT distribution, β has been shown to predict satisfiability (Mitchell, Selman, and Levesque, 1992). The average difference between the number of positive and the number of negative literals of each variable, known as Δ, has also been shown predictive for random 3-SAT formulas, especially in combination with β (Sandholm, 1996). Pennock and Stout (1996) proposed the PE-SAT predictor, which uses a clever approximation of variable dependencies to compute the expected number of satisfying assignments in $O(n^2)$ time ($O(n^3)$ for an extended version). They also showed that PE-SAT could be used during a DLL search. By pruning any node representing a subformula that PE-SAT predicts has few or no solutions, the algorithms avoids unproductive subtrees, at the risk of overlooking a satisfying assignment. By varying the pruning threshold, accuracy can be traded for decreased running time.

Unfortunately, PE-SAT-pruned DLL is not a practical algorithm, as we will see below in figure 5. Its $O(n^2)$ prediction at every node is too expensive, and it is unclear how to reuse portions of previous computations. But what about using a less expensive predictor? β and Δ are easy to compute, although they have only been shown effective on formulas drawn from the random 3-SAT distribution. Let's examine their behavior on the subformulas that a DLL algorithm generates during its search.

3 Training Data

Predicting satisfiability from β and Δ is most effective when one can compare measurements to prior results on similar formulas, so we must gather data on the probability of satisfiability of DLL subformulas with various values of β and Δ. When measuring the subformulas, we might expect the characteristics of the simpler ones to shift away from those exhibited by the original formula's distribution, so we can use the percentage of assigned variables as an additional classification statistic. Figure 1 shows the upper bound of a 99% confidence interval on the probability of satisfiability of subformulas of DLL. The data were gathered while running DLL with a variable choice heuristic similar to that used by TABLEAU (Crawford and Auton, 1996), except that the true number of unit propagations which would result from selecting each candidate variable was not computed. Variables were set first to that value that would make the greater number of their literals true. 100,000 formulas of 80 variables were used. They were drawn from the random 3-SAT distribution, with initial β values ranging uniformly from $\frac{1}{2}$ to $1\frac{1}{2}$ times the value at which 50% of the instances would be expected to be

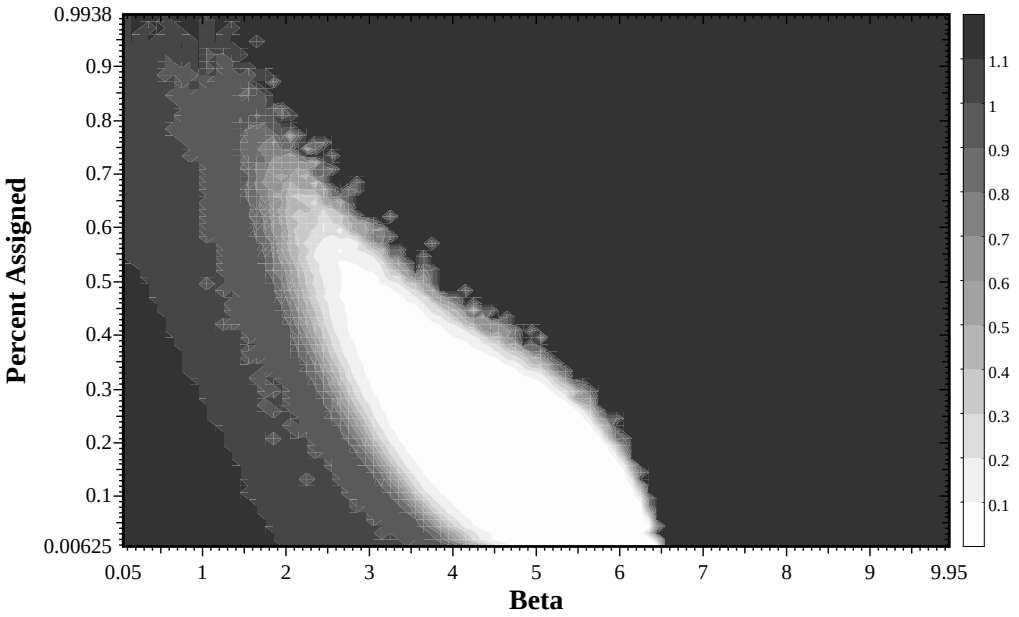

Figure 1: Upper bound on the confidence that DLL subformulas are satisfiable. 1.1 indicates no data.

satisfiable. Results were stored in a two-dimensional table with 80 entries along the percent-assigned axis (for values between 0 and 1) and 100 entries along the β axis (for values between 0 and 10). Confidence intervals were calculated asymmetrically (Lindgren and McElrath, 1959). For table entries with no formulas, we have no idea what the upper bound should be and it is plotted as 1.1.

The swath of figure 1 containing data is tilted to the left, indicating that, in general, β decreases as variables are assigned. This makes sense, since true literals eliminate clauses containing unassigned variables. The bottom portion of the plot, corresponding to few assigned variables, matches previous results concerning random 3-SAT formulas, showing a monotonic dependence on β with a steep transition from satisfiable to unsatisfiable between β values of 4 and 4.5. The upper portion of the plot also matches intuition, with DLL unable to assign a large percentage of the variables in many unsatisfiable formulas. The middle portion of the plot is interesting, and a slice taken across β with 55% of the variables assigned appears in figure 2. DLL subformulas do not simply have the smooth satisfiability profile of random 3-SAT formulas shifted to the left—the data show a perturbation

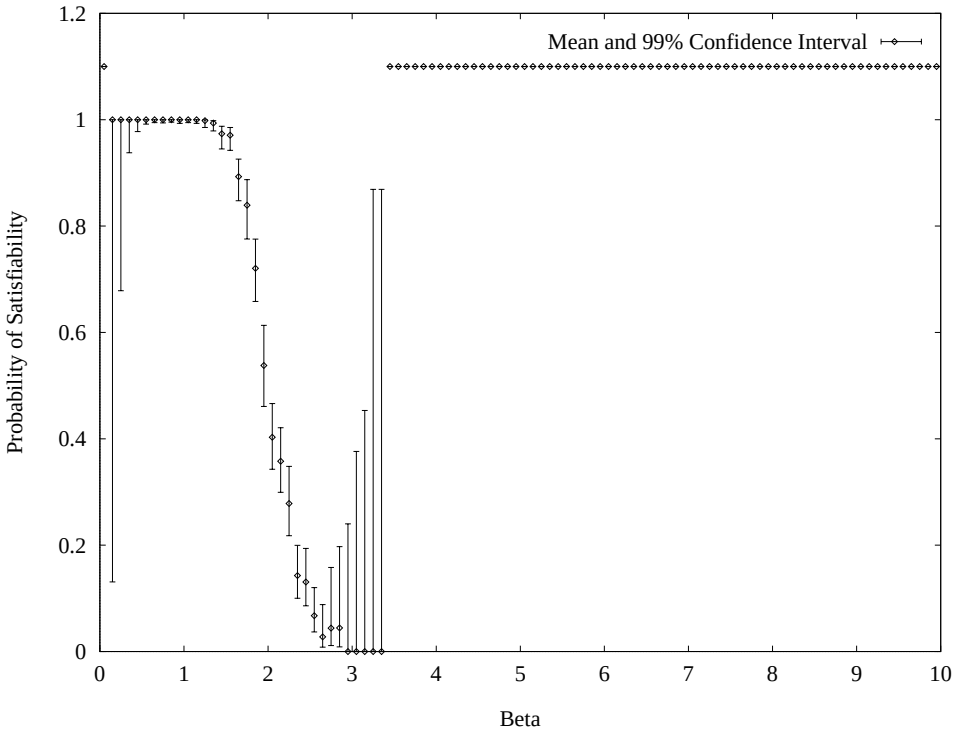

Figure 2: A slice through figure 1 when 55% of the variables are assigned. 1.1 indicates no data.

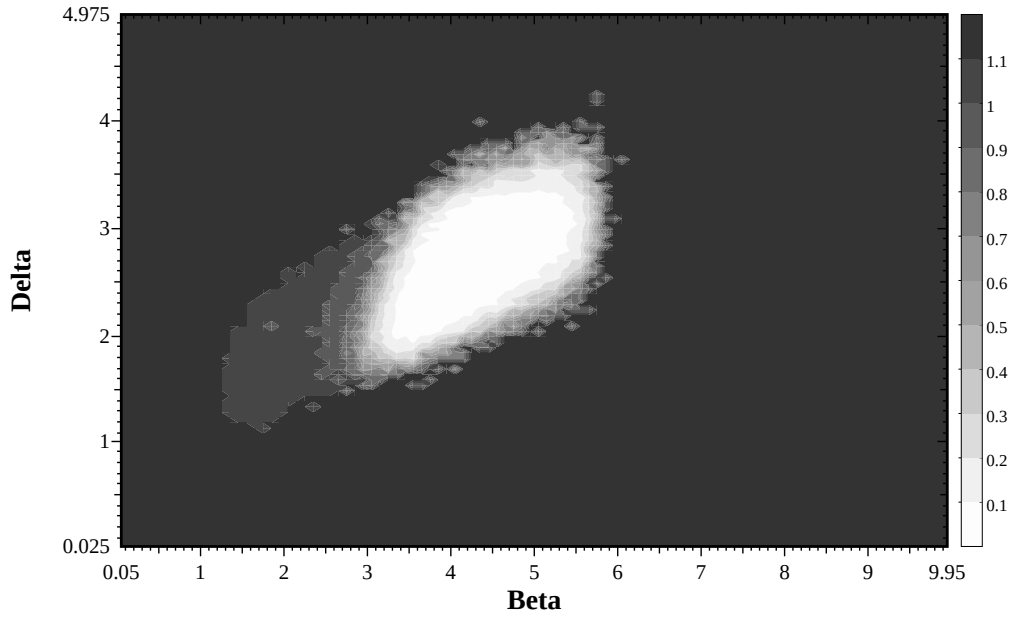

Figure 3: Upper bound on satisfiability when 20% of the variables are assigned. 1.1 indicates no data.

when β is 2.2, and a slower approach to unsatisfiability around 2.8 than one would expect. Despite these anomalies, the data in figure 1 show that there are wide ranges where β seems useful for predicting unsatisfiability.

Using Δ improves the segregation of satisfiable and unsatisfiable subformulas. Figure 3 shows a slice through results gathered from DLL using a three-dimensional table indexed by β, Δ, and the percentage of variables assigned. The Δ axis had 100 entries, for values between 0 and 5. The rightward tilt of the contours, particularly at mid-range values of β, suggests that a larger disparity between the number of positive and negative literals lowers the number of contradictions and improves the chance of finding a satisfying assignment. This agrees with the findings of Sandholm (1996) on random 3-SAT formulas.

4 Performance

Now that we have data about the satisfiability of DLL subformulas as a function of β and Δ, we can use that information for pruning the search tree. At every node, we can calculate β, Δ, and percentage of variables that

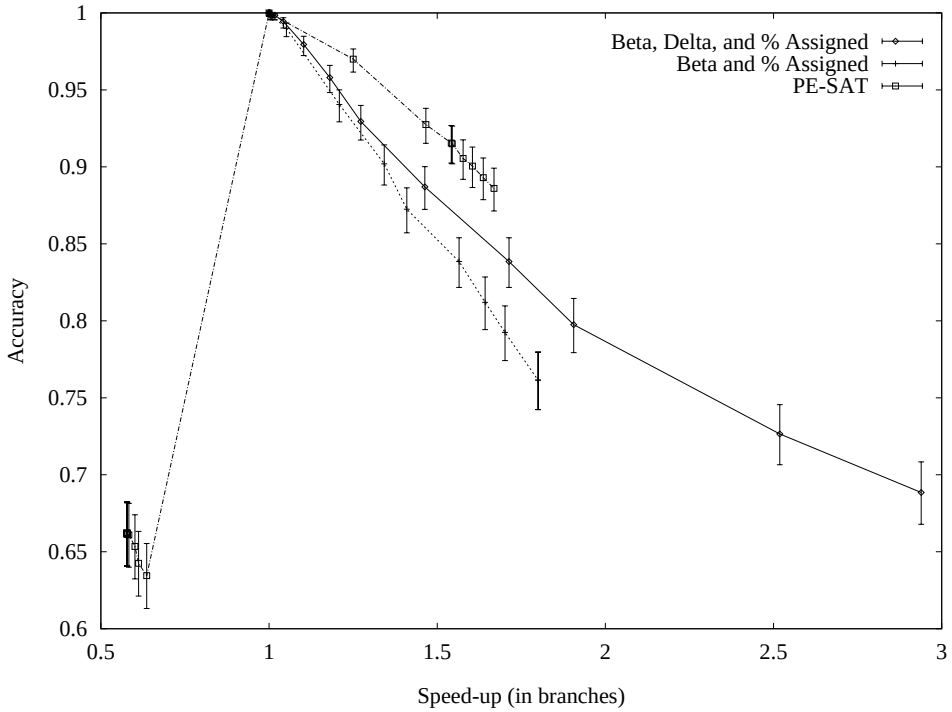

Figure 4: Performance of DLL with pruning using various predictors, measured in branches.

have been assigned, and then index into the training data. If prior experience indicates a very low upper bound on the probability that the subformula is satisfiable, we won't bother expanding the node. By adjusting the upper bound threshold, we can trade accuracy for speed.

Figure 4 shows the accuracy of DLL with pruning versus the number of nodes expanded. The number of nodes is plotted as speed-up relative to the same algorithm with no satisfiability prediction or pruning, so a values of 2 indicates half the usual number of branches in the search tree.[2] The variable- and value-selection heuristics were as described above. 2,000 hard random 3-SAT instances of 40 variables were used for testing. The β and percentage assigned, and β, Δ, and percentage assigned data were gathered from a separate collection, as described above except containing formulas of 40 variables. Error bars indicate 95% confidence intervals.

[2]Speed-up tends to be lower when measured in branches rather than nodes generated, since some implementations, such as that of Pennock and Stout (1996), appear to count each unit propagation as a node.

PE-SAT seems to be the most accurate predictor, although it abruptly exhibits pathological behavior as the threshold number of solutions rises past 4. It seems to prune every satisfiable branch, forcing DLL to explore more of the tree while avoiding any solutions, resulting in low accuracy and an enormous number of branches. These data points appear at the left of the plot, even though they result from moving the threshold past the value used to generate the points falling to the right. While less accurate, the simpler predictors have smoothly varying performance curves. More importantly, they are much easier to compute. The number of variables and clauses in each subformula can be computed by subtraction from the parent node as the formula is simplified. Since unit propagation already visits all clauses in which the recently-set variable's literal is becoming false and being eliminated, the additional effort to count the clauses that are being eliminated, which are those in which the variable's literal is become true, is only a constant factor. Therefore, β can be computed in constant time as each node. Similarly, eliminated clauses can be checked for occurrences of the literals of unbound variables, allowing Δ to be updated with constant time overhead. Given a confidence level and upper bound, some additional time can be saved by precomputing confidence intervals and pruning decisions for each table entry. One might suspect that the $O(n^2)$ running time of PE-SAT wouldn't be a serious liability in practice, since most subformulas are fairly small, but this is not the case. For 40 variable instances, all PE-SAT runs took at least 181 times as long as a plain DLL with no pruning, for 80 variables instances, at least 407 times as long, and for 120 variables instances, at least 632 times as long.

Figure 5 compares the same runs of pruned DLL to WSAT. Since WSAT does no unit propagation, a comparison based on nodes would be misleading, so we compare the algorithms on the basis of observed running time. As before, speed-up is relative to DLL without prediction or pruning. On these 40-variable instances, the plain version of DLL performed 16.3 branches per formula at around 6,000 branches per second, and WSAT performed 66–71,000 flips per second. These running times correspond to a DEC AlphaStation 500/500 (SPECint95 15.0, SPECfp95 20.4). β and Δ were computed efficiently as described above, although the variable-selection heuristics were implemented naively, taking time linear in the size of the original formula at every node. WSAT was implemented using the same efficient data structures as were shown best by Fukunaga (1997), and with attention to the details discussed by Parkes and Walser (1996). We allowed WSAT to make $0.0445287n^{2.3207}$ flips before restarting with a new random assignment. This is our fit to the reported optimal data of Parkes and Walser (1996), and it

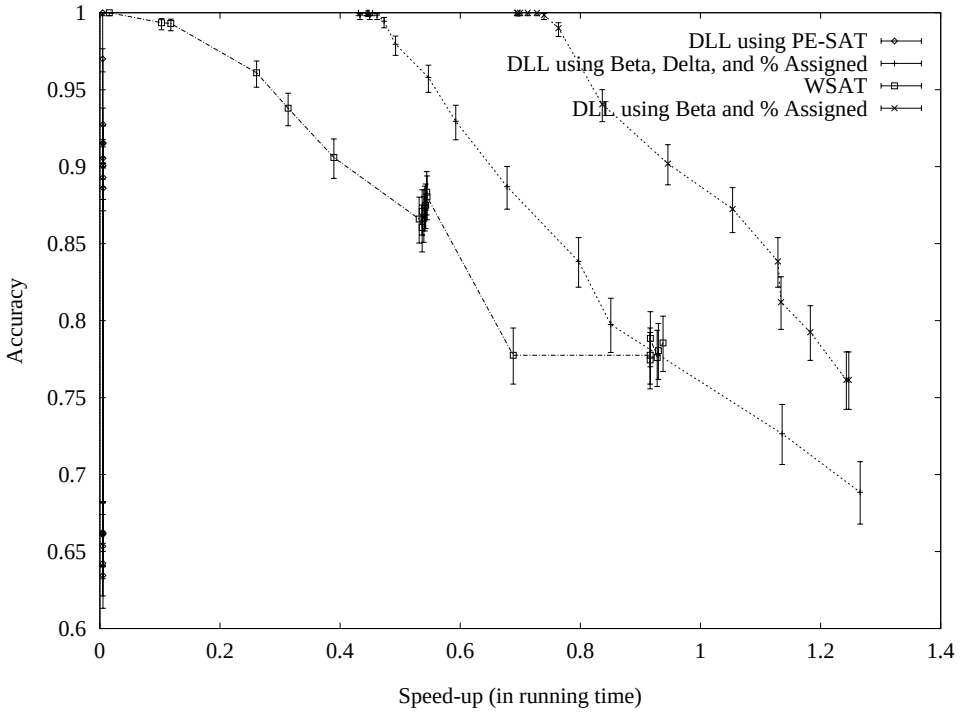

Figure 5: Performance of DLL with pruning compared to WSAT on 40-variable instances.

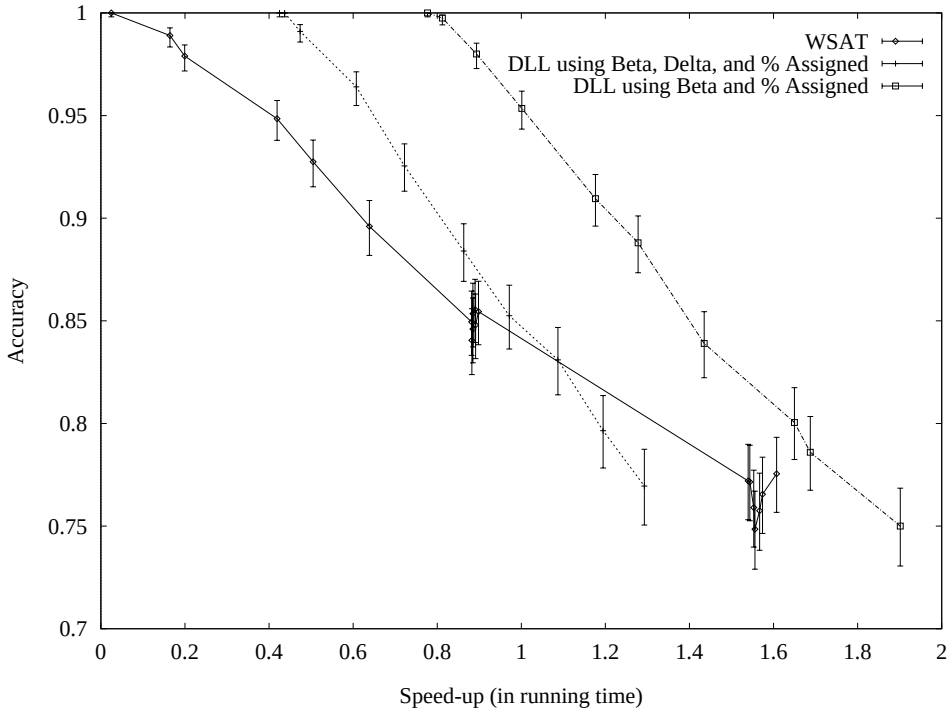

Figure 6: Performance of DLL with pruning compared to WSAT on 80-variable instances.

gave slightly better results than their recommended $0.02n^{2.39}$ flips. To handle unsatisfiable instances, we imposed a limit on the number of restarts, and varied this limit to allow a trade-off between accuracy and time.

Since calculating the predictors and checking against training data takes time, the runs of DLL using pruning must sacrifice accuracy just to equal the running time of the unadorned algorithm. The increased accuracy of prediction using Δ does not offset its additional computation time, at least in our implementation. DLL using PE-SAT lies against the left edge of the plot. The poor performance of WSAT is caused by its blind tenacity—since it can't recognize unsatisfiable formulas, it must reach its restart limit for each one. Although many satisfiable formulas are solved quickly, high accuracy requires large numbers of restarts, and this becomes a penalty to be paid on each unsatisfiable instance.

Figures 6–9 show the performance of the algorithms as the size of the formulas is increased. Separate training data was gathered for each size, except at 160 and 200 variables, where the data from 120-variable formulas

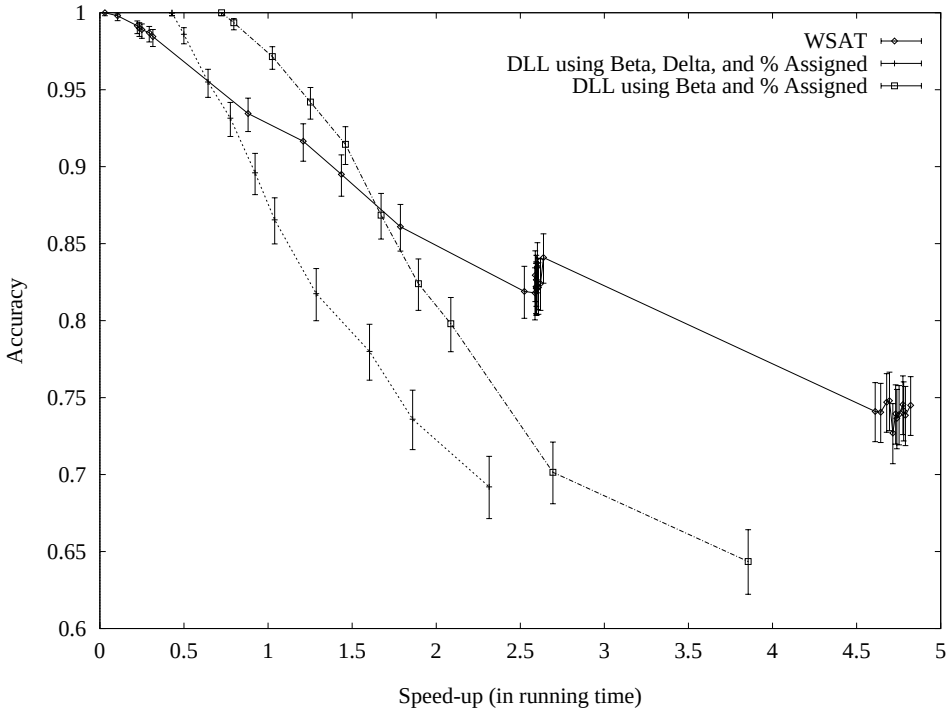

Figure 7: Performance of DLL with pruning compared to WSAT on 120-variable instances.

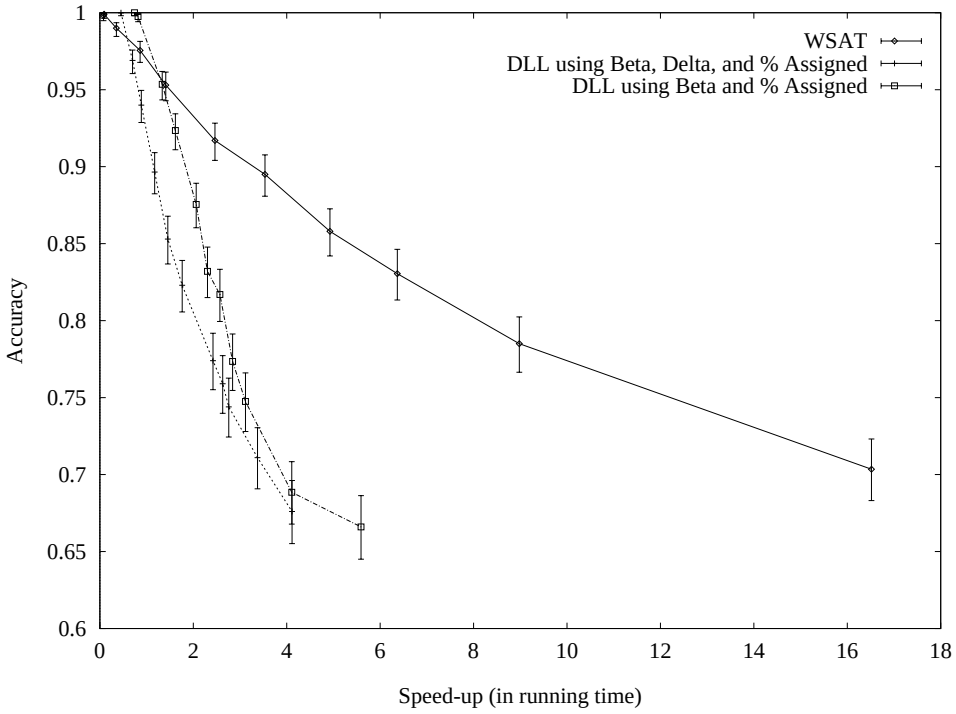

Figure 8: Performance of DLL with pruning compared to WSAT on 160-variable instances.

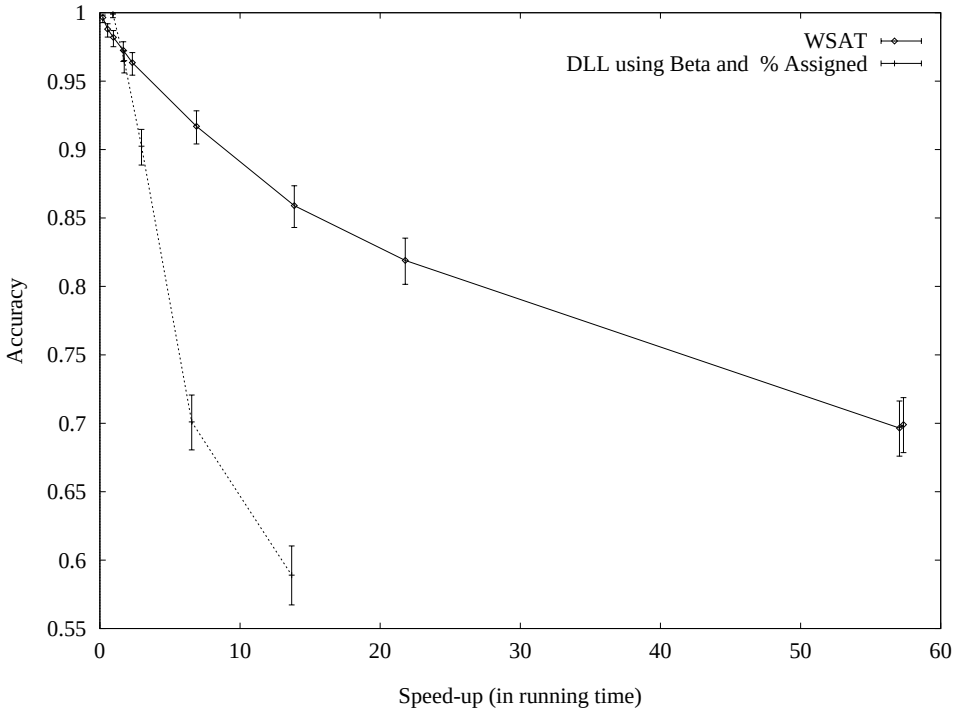

Figure 9: Performance of DLL with pruning compared to WSAT on 200-variable instances.

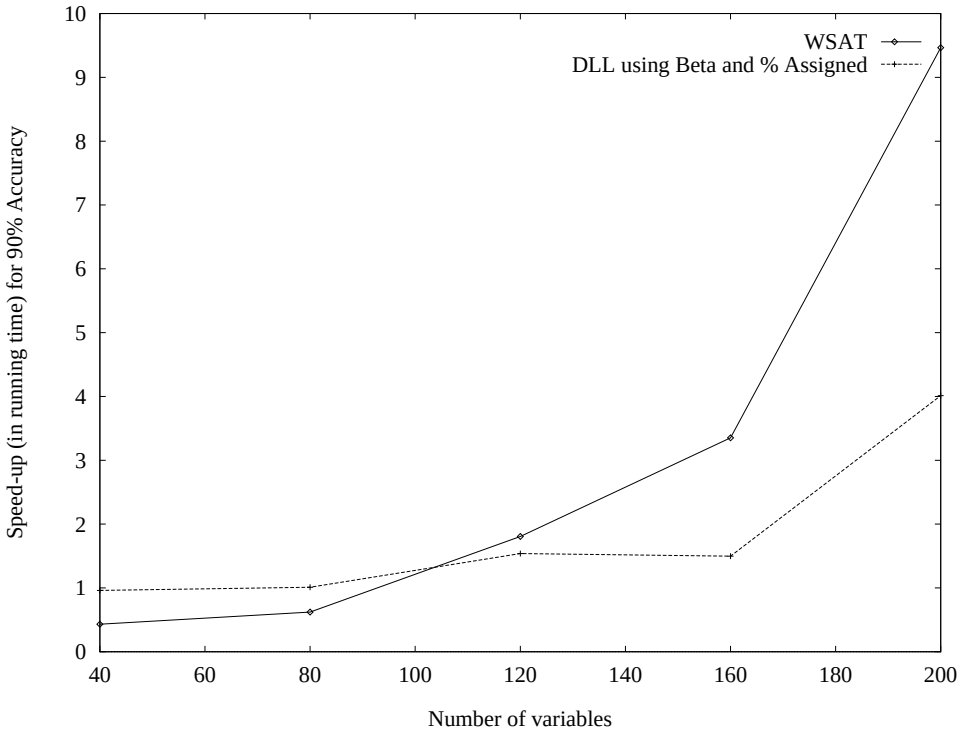

Figure 10: Speed-up of pruned DLL and WSAT relative to plain DLL for 90% accuracy.

was used. As the number of variables increases, so does the accuracy of all the algorithms. The overhead of computing the pruning predictors remains constant, as we would expect. The extra accuracy of Δ never becomes cost-effective. Although it remains slower than plain DLL for high accuracies, WSAT improves faster at lower accuracies than DLL with pruning, dominating the tree-based search for larger problems. A plot of the speed-up achieved for 90% accuracy is shown in figure 10. Unfortunately, it is not clear if these scaling results are due to poor scaling of our implementation of the variable-selection heuristics or an inherent property of the two kinds of algorithms.

As we noticed with 40-variable instances, WSAT spends most of its time on unproductive restarts for unsatisfiable formulas. When pruned DLL and WSAT spend the same amount of time for the same accuracy, as we find with 95% accuracy on 160-variable formulas, WSAT's mean time per formula is more than 6 times greater for unsatisfiable than satisfiable formulas, whereas DLL only spends 2.5 times as long on them. Although it takes longer to find

a satisfying assignment that WSAT, DLL can recognize unsatisfiable formulas and cease working on them.

Similar DLL results were obtained when using β and percentage assigned data gathered from a collection of exclusively hard random 3-SAT formulas. Preliminary experiments also showed similar performance on 80-variable instance when using training data from 40-variable instances. Using depth in the search tree rather than the percentage of variables assigned gave insignificantly worse results. Preliminary experiments have shown promising results when using pruned DLL on random 3-SAT formulas with a variety of β values. It remains to be seen how the algorithm performs on satisfiability problems representing reductions of other NP-complete problems.

If the actual satisfying assignment is not needed, preliminary experiments have shown that even greater gains in running time can be achieved by also using the predictors to guess when a subformula is satisfiable.

5 Possible Extensions

We have shown that traditional measures of satisfiability for random 3-SAT formulas can be extended for use with the subformulas occurring in the DLL search tree. By using them for speculative pruning, we developed an incomplete algorithm for boolean satisfiability that does not organize its search space as a graph. The basic framework of a pruned DLL is simple, and if a new predictive statistic were developed for subformulas, it could easily be incorporated into the algorithm. Increased accuracy must always be balanced against increased running time, however, as we saw with Δ. One possibility would be to compute expensive estimates only occasionally.

Predictors for DLL subformulas can also be used in other algorithms. One could imagine variable selection heuristics that gauge the promise of subtrees by probing a constant number of paths and estimating the satisfiability of the deeper nodes. Or the predictors could be used for a search method based on heuristic probing (Bresina, 1996).

Although we have also shown the trade-off between accuracy and speed for WSAT and a pruning DLL, further analysis is needed to ensure that these trade-offs hold for individual instances. With WSAT, it is easy to expend more running time on an individual instance by starting with yet another random assignment. With DLL, an iterative relaxation of the probability threshold could be used, although it is not immediately clear how smoothly the probabilities change as one descends the search tree of a particular instance.

6 Acknowledgments

This material is based upon work supported in part by the National Science Foundation under Grant Nos. IRI-9350192, and IRI-9618848.

References

Bresina, John L. 1996. Heuristic-biased stochastic sampling. In *Proceedings of AAAI-96*, pages 271–278. AAAI Press/MIT Press.

Cook, Stephen A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on the Theory of Computing*, pages 151–158. Association for Computing Machinery.

Crawford, James M. and Larry D. Auton. 1996. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81:31–57, March.

Davis, Martin, George Logemann, and Donald Loveland. 1962. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397.

Davis, Martin and Hilary Putnam. 1960. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7(3):201–215.

Even, S., A. Itai, and A. Shamir. 1976. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5:691–703.

Fukunaga, Alex S. 1997. Variable-selection heuristics in local search for SAT. In *Proceedings of AAAI-97*, pages 275–280. AAAI Press/MIT Press.

Garey, Michael R. and David S. Johnson. 1991. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York.

Lindgren, Bernard W. and G. W. McElrath. 1959. *Introduction to Probability and Statistics*. Macmillan.

Mitchell, David, Bart Selman, and Hector Levesque. 1992. Hard and easy distributions of SAT problems. In *Proceedings of AAAI-92*, pages 459–465, San Jose, CA, July. AAAI Press/MIT Press.

Parkes, Andrew J. and Joachim P. Walser. 1996. Tuning local search for satisfiability testing. In *Proceedings of AAAI-96*, pages 356–362. AAAI Press/MIT Press.

Pennock, David M. and Quentin F. Stout. 1996. Exploiting a theory of phase transitions in three-satisfiability problems. In *Proceedings of AAAI-96*, pages 253–258. AAAI Press/MIT Press.

Sandholm, Tuomas W. 1996. A second order parameter for 3SAT. In *Proceedings of AAAI-96*, pages 259–265. AAAI Press/MIT Press.

Selman, Bart, Henry A. Kautz, and Bram Cohen. 1994. Noise strategies for improving local search. In *Proceedings of AAAI-94*, pages 337–343. AAAI Press/MIT Press.