
Simulated Annealing for Hard Satisfiability Problems

William M. Spears

AI Center

Naval Research Laboratory

Washington, D.C. 20375-5320

E-mail: SPEARS@AIC.NRL.NAVY.MIL

Abstract

Satisfiability (SAT) refers to the task of finding a truth assignment that makes an arbitrary boolean expression true. This paper compares a simulated annealing algorithm (SASAT) with GSAT (Selman *et al.*, 1992), a greedy algorithm for solving satisfiability problems. GSAT can solve problem instances that are extremely difficult for traditional satisfiability algorithms. Results suggest that SASAT scales up better as the number of variables increases, solving at least as many hard SAT problems with less effort. The paper then presents an ablation study that helps to explain the relative advantage of SASAT over GSAT. Next, an improvement to the basic SASAT algorithm is examined, based on a random walk implemented in GSAT (Selman *et al.*, 1993). Finally, we examine the performance of SASAT on a test suite of satisfiability problems produced by the 1993 DIMACS challenge.

1 Introduction

Satisfiability (SAT) refers to the task of finding a truth assignment that makes an arbitrary boolean expression true. For example, the boolean expression $a \ \& \ b$ is true if and only if the boolean variables a and b are true. Satisfiability is of interest to the logic, operations research, and computational complexity communities. Due to the emphasis of the logic community, traditional satisfiability algorithms tend to be sound and complete.¹ However, Selman *et al.* (1992) point out that there exists a class of satisfiability problems that are extremely hard for these algorithms. Their response has been to create a greedy algorithm (GSAT) that is sound, yet incomplete (i.e., there is no guarantee that GSAT will find a satisfying assignment if one exists). The advantage of

¹ Soundness means that any solution found must be correct. Completeness means that a solution must be found if one exists.

GSAT is that it can often solve problems that are extremely difficult for the traditional algorithms.

Other recent work has also concentrated on sound but incomplete algorithms for satisfiability (De Jong and Spears, 1989; Spears, 1990; Young and Reel, 1990; Gu, 1992). However, comparisons between the algorithms have been difficult to perform, due to a lack of agreement on what constitutes a reasonable test set of problems. One nice feature of the Selman *et al.* (1992) paper is that a class of hard problems is very precisely defined. In this paper we compare GSAT with a novel simulated annealing approach (SASAT) on that class of hard problems. The results suggest that SASAT solves at least as many problems with much less effort. Next, we modify the simulated annealing algorithm, to illustrate why SASAT outperforms GSAT. Thirdly, the paper examines an enhancement to SASAT, based on a random walk feature described in Selman *et al.* (1993). Finally, we examine the performance of SASAT on a test suite of satisfiability problems produced by the 1993 DIMACS challenge. First, however, we provide an overview of GSAT and introduce the simulated annealing algorithm.

2 GSAT and SASAT

GSAT assumes that the boolean expressions are in conjunctive normal form (CNF).² After generating a random truth assignment, it tries new assignments by flipping the truth assignment of a boolean variable that leads to the largest δ (increase) in the number of true clauses. GSAT is greedy because it always tries to increase the number of true clauses. If it is unable to do this it will make a "sideways" move (i.e., change the truth assignment of a variable although the number of true clauses remains constant). GSAT can make a "backwards" move (decrease the number of true clauses), but only if other moves are not available. Furthermore, it can not make two backwards moves in a row, since the backwards move will guarantee that it is possible to increase the number of true clauses in the next move. The algorithm for GSAT is presented in Figure 1.

```
Procedure GSAT;
Input: A set of clauses, MAX_FLIPS, and MAX_TRIES;
Output: A satisfying truth assignment of the clauses, if found;

for i = 1 to MAX_TRIES {
  T = a random truth assignment;
  for j = 1 to MAX_FLIPS {
    if T satisfies the clauses then return T;

    flip a variable that results in the largest  $\delta$ 
    (increase) in the number of clauses made true;

    T = the new assignment after the flip is made;
  } }
```

Figure 1: The GSAT algorithm.

Recently, Spears (1993) showed that a Hopfield-style neural network with simulated annealing outperforms GSAT on hard satisfiability problems. The neural network algorithm makes no assumptions about the form of the boolean expression. By

² CNF refers to boolean expressions that are a conjunction of clauses, each clause being a disjunction of negated or non-negated boolean variables.

specializing to CNF expressions, the neural network can be dropped, resulting in a simulated annealing algorithm we call SASAT. The algorithm for SASAT is presented in Figure 2.

```

Procedure SASAT;
Input: A set of clauses, MAX_TRIES, MAX_TEMP, and MIN_TEMP;
Output: A satisfying truth assignment of the clauses, if found;

i = 0; tries = 0;
loop {
  i++;
  T = a random truth assignment; j = 0;
  loop {
    if T satisfies the clauses then return T;

    temperature = MAX_TEMP * e-j * decay_rate;
    if (temperature < MIN_TEMP) then exit loop;

    for v = 1 to the number of variables V in the clauses {
      Compute the increase (decrease)  $\delta$  in the number of
      clauses made true if v were flipped;

      flip variable v with probability defined by the
      logistic function:  $\frac{1}{1 + e^{-\frac{\delta}{\text{temperature}}}}$  ;

      T = the new assignment if the flip is made;
    }
    j++; tries++;
    if (MAX_TRIES = tries) terminate algorithm;
  } }

```

Figure 2: The SASAT algorithm.

SASAT has a control structure very similar to the control structure for GSAT. The outermost loop variable i is analogous to the variable i in GSAT. Thus i reflects the number of independent attempts to solve the problem. Each time i is incremented the algorithm randomly generates a truth assignment for the boolean variables and the temperature is set to MAX_TEMP. The inner loop, indexed by j , tries new assignments by probabilistically flipping each boolean variable individually, based on the improvement (δ) this flip would bring. If the improvement is positive the flip is likely to be performed. If the improvement is negative the flip is unlikely to be performed. SASAT differs from GSAT in that it can make arbitrary sequences of "backwards" moves, which is a necessary feature for escaping local optima in the search space.

The probabilistic moves (flips) are determined using the standard logistic function for simulated annealing (see Figure 2). When the temperature is high the moves are almost random, and when the temperature is low SASAT is similar to GSAT. The inner loop controls the annealing schedule (i.e., the rate at which the temperature drops from MAX_TEMP to MIN_TEMP). Note that as j increases, the temperature slowly decreases according to the decay rate in the following fashion:

$$\text{temperature} = \text{MAX_TEMP} * e^{-j * \text{decay_rate}}$$

Once the minimum temperature is reached, i is incremented, and SASAT tries again to

solve the problem. However, a good heuristic is to reduce the decay rate before making another attempt, in order to perform more flips during the next attempt. We used our experience with the neural network algorithm (Spears, 1993) to set the decay rate to be:

$$decay_rate = \frac{1}{i * V}$$

where V is the number of variables. Thus, each time i is incremented the decay rate decreases. Also, the decay rate is dependent on the number of variables in the problem to be solved. SASAT will use smaller decay rates on problems with more variables, because reducing the temperature more slowly is a good heuristic for larger problems. Finally, we used our prior experience with the neural network algorithm to set MAX_TEMP to 0.3 and MIN_TEMP to 0.01.

Clearly these choices in parameters will entail certain tradeoffs. For a given setting of MAX_TRIES, reducing MIN_TEMP and/or increasing MAX_TEMP will allow more tries to be made per independent attempt, thus decreasing the number of times that i can be incremented before the MAX_TRIES cutoff is reached. A similar situation occurs if we decrease or increase the decay rate. Thus, by increasing the temperature range (or decreasing the decay rate) we reduce the number of independent attempts, but search more thoroughly during each attempt. The situation is reversed if one decreases the temperature range (or increases the decay rate). Unfortunately it is not at all clear whether it is generally better to make more independent attempts, or to search more thoroughly during each attempt. Although we have made some effort to find a reasonable balance between the number of attempts and the amount of search within an attempt, we make no claims to optimality in this paper.

2.1 Implementation Details

As will be discussed in the Section 3, the bulk of the computation in both SASAT and GSAT lie in performing flips and computing δ s. In order to facilitate this discussion, we now describe the data structures and algorithms used by SASAT to perform these operations. The two most important data structures in SASAT are called the clause list (c_list) and the variable list (v_list). Both of these are essentially two dimensional arrays, although each row can have a different number of entries. The c_list is used to maintain information about clauses: $c_list[i]$ maintains information about clause i . The first entry in $c_list[i]$ (i.e., $c_list[i][0]$) is used to indicate how many of the literals are true in the clause, for some given assignment of the boolean variables.³ The remainder of $c_list[i]$ has a list of variables contained in that clause. For ease of programming, a negative entry means the variable is negated in clause i . We allow each row to have a different number of entries to allow for boolean expressions that have clauses of differing lengths.

The v_list is used to maintain information about variables: $v_list[i]$ maintains information about variable i . The first entry in $v_list[i]$ (i.e., $v_list[i][0]$) is the number of clauses that the variable i is in, while the remainder of $v_list[i]$ has a list of the clauses that the variable is in. Again, we use a negative entry to denote when a variable is negated in some clause. Again, each row can have a different number of entries because variables need not appear in the same number of clauses.

To illustrate these data structures consider the following conjunctive normal form boolean expression (i.e., a conjunction of disjunctions) consisting of three literals per

³ A literal is a negated or non-negated boolean variable.

clause, four clauses, and four variables (x_1, x_2, x_3 , and x_4):

$$(\overline{x_2} \text{ or } \overline{x_3} \text{ or } \overline{x_4}) \& (x_1 \text{ or } \overline{x_2} \text{ or } \overline{x_3}) \& (\overline{x_3} \text{ or } x_1 \text{ or } \overline{x_2}) \& (x_4 \text{ or } x_3 \text{ or } \overline{x_2})$$

Suppose we have not yet attempted to make an assignment to the variables. Then the data structures would appear as in Tables 1 and 2 below. Table 1 illustrates the c_list for the above boolean expression, while Table 2 illustrates the v_list . Due to our choice of problem each clause happens to have the same length (again, this is not necessary), but note that each variable does not appear in the same number of clauses. Note also that the number of true literals for each clause (in Table 1) is 0, since we have not yet made an assignment to the variables. Now suppose we happen to randomly initialize all four variables to true. Then the c_list is used to compute how many true literals are in each clause. The result is shown in Table 3.

Clause	# of true literals	variables in the clause
clause 1	0	{-2 -3 -4}
clause 2	0	{1 -2 -3}
clause 3	0	{-3 1 -2}
clause 4	0	{4 3 -2}

Table 1: The clause list (with no truth assignment)

Variable	# of clauses variable is in	clauses variable is in
x_1	2	{2 3}
x_2	3	{-1 -2 -3 -4}
x_3	4	{-1 -2 -3 4}
x_4	3	{-1 4}

Table 2: The variable list

Clause	# of true literals	variables in the clause
clause 1	0	{-2 -3 -4}
clause 2	1	{1 -2 -3}
clause 3	1	{-3 1 -2}
clause 4	2	{4 3 -2}

Table 3: Clause list (with all variables true)

Clearly we have not yet satisfied the boolean expression, since the first clause does not have any true literals. Thus we need to consider flipping variables. Suppose we randomly choose x_4 as a candidate for flipping. To compute the δ we use the v_list to note that x_4 is contained in the first clause as a negated variable, and contained in the fourth clause as a non-negated variable. Since x_4 is currently true, flipping it to false will thus increase the number of true literals in the first clause by one, and decrease the number of true literals by one in the fourth clause. This will make the first clause true, since it currently does not have any true literals. However, the truth value of the fourth clause remains unaffected, since it already has two true literals. Thus, the result of

flipping x_4 will be to increase the number of satisfied clauses from three to four (i.e., $\delta = 1$). Now, if we actually perform this flip, the c_list is updated as shown in Table 4. SASAT would now notice that all four clauses are in fact satisfied and would terminate successfully.

Clause	# of true literals	variables in the clause
clause 1	1	{-2 -3 -4}
clause 2	1	{1 -2 -3}
clause 3	1	{-3 1 -2}
clause 4	1	{4 3 -2}

Table 4: Clause list (with x_4 false, and the rest true)

Before we close this section, a note on the computational cost of a δ calculation is in order. If we consider the example given above we note that the factor most heavily influencing the δ computation is the number of clauses that a variable is in. Thus, if we fix the number of variables and increase (decrease) the number of clauses, we increase (decrease) in a fairly linear fashion the cost of computing a δ . Similarly, if we keep the ratio of the number of clauses to the number of variables constant, the cost of the δ computation is largely unaffected. We will use this observation to explain some of the results presented later in this paper.

3 Comparison, Experiments, and Results

In comparing two (or more) algorithms, one difficult choice is in the selection of problem instances to solve. Since traditional satisfiability algorithms already work well on many problems, it is useful to consider those problems where the traditional algorithms run into difficulty. Furthermore, to avoid the risk of overfitting an algorithm to a particular problem, it is important to either select a large number of problems or to draw problems from a particular problem class (or distribution). Fortunately, classes of problems that are difficult for traditional satisfiability algorithms have already been identified. In this paper we will concentrate on one such class, a fixed clause-length model referred to as *Random L-SAT* (Mitchell *et al.*, 1992). The performance of GSAT on Random L-SAT problems has already been reported in Selman *et al.* (1992). Although we could not obtain the specific problems used in their experiments, we generated random problems using their random problem generator.

Another difficult choice is in *how* to measure and compare the performance of the two algorithms. Clearly, one important measure is the average amount of computation performed by both algorithms. For example, measuring the number of flips performed by each algorithm would be one obvious choice for comparison. However, due to the similarity of the two algorithms, we can be even more careful in our comparison. For both GSAT and SASAT, the bulk of the computation lies not only in the manipulation of the data structures when a flip is made (see Figures 1 and 2), but also in the computation of the δ s. Although both a flip and the computation of each δ can be performed fairly efficiently through the use of carefully chosen data structures, the complexity increases with the number of clauses. Other steps in the algorithms, such as calls to a random number generator, the check for termination, the computation of the logistic function, or the determination of the best variable (in GSAT) are less computationally intensive.

If we are to use flips for our measure of comparison, then, we would have to feel comfortable that both GSAT and SASAT perform roughly the same number of δ

computations per flip. Selman *et al.* (1992) did not report the number of δ s computed by GSAT, but they can be estimated from the reported number of flips. At first blush the number of δ s computed would appear to be the number of flips multiplied by the number of variables V , since an obvious way to find the best variable is to compute the δ for each variable, selecting a variable with the highest δ . However, as Selman points out (personal communication), after each flip it is only necessary to compute the δ s of those variables that share one or more clauses with the flipped variable. Put another way, if a variable does *not* share a clause with the flipped variable, nothing has changed in any clause that variable is in, and the δ for that variable need not be recomputed.

For the purposes of illustration we will present an example of this, using the previous boolean expression and the SASAT data structures. Suppose x_1 is false, while x_2 , x_3 , and x_4 are true. Then the clause list is as shown in Table 5. Note that only the last clause is satisfied. Let us now calculate the δ that would occur if x_1 were made true. Note that the only effect is to satisfy the second and third clauses. Thus δ_{x_1} is 2. Similarly we can compute that δ_{x_2} is 3, δ_{x_3} is 3, and δ_{x_4} is 1. Suppose for the sake of illustration that we do indeed flip x_1 to true. Then the clause list appears as in Table 6. If we recompute the δ s we get δ_{x_1} is -2, δ_{x_2} is 1, δ_{x_3} is 1, and δ_{x_4} is 1. Note that δ_{x_1} is of course the negation of what it was earlier (since we flipped x_1). Note also that the δ s for x_2 and x_3 have changed. However, δ_{x_4} has not changed, because it does not share any clauses with x_1 . Thus it really was unnecessary to recompute δ_{x_4} .

Clause	# of true literals	variables in the clause
clause 1	0	{-2 -3 -4}
clause 2	0	{1 -2 -3}
clause 3	0	{-3 1 -2}
clause 4	2	{4 3 -2}

Table 5: Clause list (with x_1 false, and the rest true)

Clause	# of true literals	variables in the clause
clause 1	0	{-2 -3 -4}
clause 2	1	{1 -2 -3}
clause 3	1	{-3 1 -2}
clause 4	2	{4 3 -2}

Table 6: Clause list (with all variables true)

The random L-SAT problem generator creates random problems in conjunctive normal form subject to three parameters: the number of variables V , the number of clauses C , and the number of literals per clause L . Each clause is generated by selecting L of the V variables uniformly randomly, negating each variable with probability 50%. Let R denote the ratio of the number of clauses to the number of variables (C/V). Each clause contains L literals, so a random problem will contain $L * C = L * R * V$ literals.

Now suppose some variable v is in c clauses. Then this means that v and \bar{v} occur a total of c times in the boolean expression, which accounts for c literals.⁴ If we continue

⁴ In this paper L will be very small compared to V . Under these conditions it is very unlikely to have the same variable occur more than once in a clause, so we can safely ignore that possibility.

in this fashion for all variables, counting the number of clauses each variable is in, we will account for all $L * R * V$ literals. Thus the V variables are in a total of $L * R * V$ clauses. Since there are only V variables, and they are chosen uniformly, we know that a variable occurs (on average) in $L * R$ clauses (as a nice example of this, note how each variable occurs in $L * R = 3$ clauses in Table 2). Finally, since each of the $L * R$ clauses contains $L - 1$ other literals, we can conclude that a given variable shares clauses with at most $L * R * (L - 1)$ other variables. This is an upper bound since the other variables need not be unique. However, uniqueness becomes more and more likely as V increases, and this upper bound is in fact a good estimate for the problems encountered in this paper.⁵

Thus far the analysis has only been with respect to the problem generator. However, since GSAT makes use of the observation that it is only necessary to compute the δ s of those variables that share one or more clauses with the last flipped variable, we can also conclude that at most $L * R * (L - 1)$ δ s are computed for every flip in GSAT. Although GSAT was not available when this paper was written, we have recently acquired GSAT and preliminary experiments do indeed confirm that this upper bound is in fact a reasonable estimate.

Unfortunately, an estimate of the number of δ computations per flip in SASAT is not available analytically, and thus the number must be determined experimentally. In general, the number of δ computations per flip will differ from the number computed for GSAT. Due to that fact, we consider it important to report both the number of δ s computed and the number of flips performed by GSAT and SASAT. As it turns out, for many of the problems considered in this paper, GSAT and SASAT do in fact perform roughly the same number of δ computations per flip. When this occurs we will find it convenient to ignore δ s and concentrate primarily on flips. However, this will not hold on other problems, and in such cases more insight is gained by considering both δ and flips.

It might be supposed that this whole issue could be resolved by simply concentrating on CPU time and by measuring both the total number of flips, and the number of flips that GSAT and SASAT make per unit of time (e.g., flips per second). After all, the amount of work done per flip is precisely captured by a statistic measuring the number of flips per second that each algorithm can perform (on a given problem). This statement is true if we limit ourselves to the current implementations of GSAT and SASAT. What is interesting, however, is that due to the similarity of the two algorithms, GSAT can be written using the data structures of SASAT, and SASAT can be written using the data structures of GSAT. Thus, both algorithms can be written using essentially the same code for computing δ s and for performing flips. Under these circumstances the number of δ s computed per second will be the same, and the key difference is in how many δ s are performed per flip for each algorithm. It is for this reason that we consider it important to concentrate on both δ s and flips in our general comparison, thus de-emphasizing the particular implementations used.

To summarize, since GSAT and SASAT can be implemented using identical data structures and the same code to compute δ s and perform flips, it is important to concentrate on both of these factors in our comparison of the two algorithms. However, it will also be of interest to compare SASAT with other algorithms that are not at all similar to SASAT. Under these circumstances it is much harder to make implementation independent comparisons, and CPU time becomes the only reasonable comparison statistic. For this reason we also report the number of flips performed by SASAT per

⁵ If we let $N = L * R * (L - 1)$ the probability that the N variables are unique is $(V/V) * ((V - 1)/V) \cdots ((V - N + 1)/V)$, which approaches 1 for large V .

second and the CPU time of SASAT for all results presented in this paper.

3.1 Experiments and Results

As mentioned earlier, in this paper we will concentrate on the fixed clause-length problems referred to as Random L-SAT problems (Mitchell *et al.*, 1992). One interesting feature of the Random L-SAT problems is that, when L is 3, the hardest problems (for traditional algorithms) occur where the clause to variable ratio R is roughly 4.25. Furthermore, when R is 4.25, roughly 50% of the random problems appear to be satisfiable.⁶ Since we are generating random problems from this distribution, it is insufficient to simply report the average number of δ s and flips required to satisfy those problems that were actually satisfied. This is because different algorithms may actually solve a different percentage of the satisfiable problems. In order to have a more meaningful comparison, then, it is important to report the percentage of problems satisfied, as well as the amount of effort required to satisfy them.

Following Selman *et al.* (1992), we generated random 3-SAT problems ranging from 100 to 500 variables, where R is 4.25. All results are averaged over 100 random instances (problems) for each choice of the number of variables. We monitored the number of δ s computed and flips performed by SASAT, and estimated the number of δ s computed by GSAT, based on the results in Selman *et al.* (1992). Since R is 4.25, roughly $3 * 4.25 * 2 = 25.5$ δ s are computed for every flip in GSAT, and we will use this result to estimate the number of δ s computed by GSAT. We also present the percentage of problems satisfied by SASAT. The percentages of problems satisfied by GSAT are not reported, however Selman (personal communication) states that GSAT satisfies roughly 20% - 33% of the 500 variable problems. We will assume that GSAT satisfies roughly 50% of the easier problems. The timing results for GSAT are taken from Selman *et al.* (1992) and Selman and Kautz (1993a).

V	C	δ s	Flips	%	MAX_FLIPS	Time
100	425	541,875	21,250	~50%	500	.1 min
200	850	12,673,500	497,000	NR	2,000	2.8 min
300	1275	35,465,400	1,390,800	NR	6,000	12 min
400	1700	89,943,600	3,527,200	NR	8,000	34 min
500	2125	253,929,000	9,958,000	20-33%	10,000	96 min

Table 7: GSAT on hard problems.⁷

Tables 7 and 8 present the percentage (%) of problems actually satisfied by the algorithms. They also give the number of δ s computed and flips performed, averaged over the problems that were satisfied. Finally, we also present the average number of independent attempts i made by SASAT before a solution was found. Given all this data, how do we compare the two algorithms? Should we use flips, δ s, or some combination of the two? Fortunately we can finesse this decision. As mentioned before, due to clever

⁶ Crawford and Auton (1993) believe that a better estimate is $4.25 + (6.21 / V)$.

⁷ NR means that this datum has not been reported. MAX_TRIES is also not reported, but is at least $10(\text{Flips} / \text{MAX_FLIPS})$. Selman *et al.* (1992) report in their table that they used 2150 clauses, yet they also state that they use $4.25V$ clauses on the harder problems. We follow the $4.25V$ guideline.

V	C	δs	Flips	%	i	MAX_TRIES	Time
100	425	581,400	31,863	58/100	6.1	200,000	.2 min
200	850	7,735,000	396,341	44/100	10.6	400,000	3 min
300	1275	37,474,500	1,924,040	48/100	17.2	800,000	13 min
400	1700	42,951,600	2,269,500	45/100	13.9	1,000,000	15 min
500	2125	86,680,500	4,438,820	41/100	15.6	1,600,000	30 min

Table 8: SASAT on hard problems.

design, GSAT computes roughly 25.5 δs for each flip, for this particular class of problems. Interestingly, if we compare the ratio of δs to flips in SASAT we see a similar pattern. For all choices of the number of variables, the ratio is roughly 20. Although not a result of design, this rather fortuitous ratio allows us to assume that both GSAT and SASAT do roughly the same amount of work per flip. For this reason we will concentrate on comparing the two algorithms on the percentage of problems satisfied and the average number of flips required to satisfy those problems.

If we examine the timing results (the last column of the table) we can also compute the number of flips accomplished per second by SASAT. Interestingly, this computation yields roughly 2500 flips per second, regardless of the number of variables. This is very likely due to the fact that, as mentioned above, the δ per flip ratio is roughly the same for all choices of the number of variables. Recall that the amount of work done per δ computation is heavily influenced by the number of clauses that a variable is in (see Section 2.1). Since this number is simply $L * R$ and both L and R are fixed, the number of δs (and flips) accomplished per second remains relatively constant as V increases.

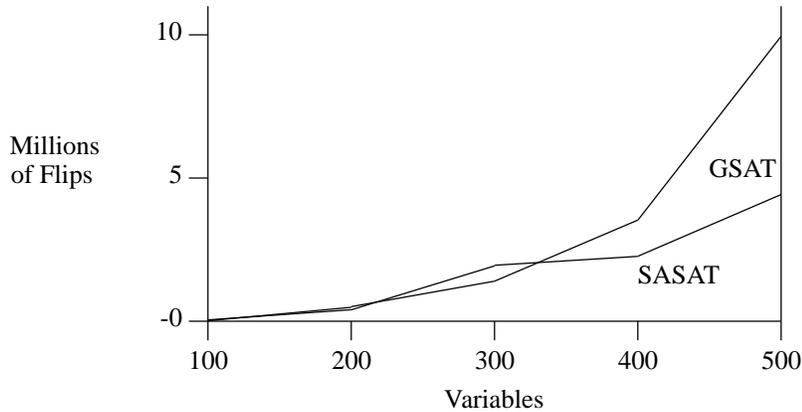


Figure 3: Comparison of GSAT and SASAT.

Figure 3 graphs the number of flips for both algorithms. In terms of flips, although GSAT may have some advantage on the easier problems, SASAT appears to scale better. A comparison of percentages is harder. The percentage of problems solved by GSAT was not strictly monitored. Also, when the number of variables was high the results for

GSAT were averaged over only 10 satisfied problems. However, in general SASAT appears to solve a higher percentage of problems with fewer flips.⁸

3.2 Distributions of Results

One of the difficulties in dealing with the above 400 and 500 variable problems is that there are no known techniques for practically determining which of the problem instances are in fact satisfiable. Thus, the results given above could be sensitive to the choices of cutoffs for the algorithms. For example GSAT is run with specific choices for the cutoff parameters MAX_TRIES and MAX_FLIPS. SASAT is run with specific choices for the parameters MAX_TEMP, MIN_TEMP, and MAX_TRIES. With the parameters set as given above, SASAT appears to solve a higher percentage of problems with less work. However, it is unlikely that SASAT satisfied all the satisfiable problems. Thus, it is conceivable that the situation could reverse if the cutoffs were increased to such an extent that both algorithms solved more (or all) of the problems. In other words, the remaining unsatisfied but satisfiable problems *could* be much more difficult for SASAT than GSAT.

Thus, if we don't know which problems are satisfiable, conclusions will necessarily be tentative. An alternative would be to solve only problems with less than 300 variables, since the satisfiable problems can be determined by sound and complete techniques. Unfortunately, our experience has shown that is also hard to draw any conclusions about the behavior of algorithms at 500 variables from their behavior at 300 variables. In either case we would have to draw tentative conclusions.

In other words, graphs of mean performance can be misleading in those cases where it cannot be known if all solvable problems have in fact been solved (Figure 13 later in this paper will provide a good example). Since it is impossible to completely resolve this issue with the current state of sound and complete algorithms, the best we can do is provide the distribution of problems actually satisfied by SASAT. This distribution graphs the number of problems solved within a certain amount of work. Although not a complete distribution (i.e., it is very likely to not include some satisfiable problems), this distribution will be helpful in comparing other algorithms with SASAT in the future. For example, although we could not use these distributions to help us compare SASAT with GSAT (since the distributions for GSAT are not available), we will find them very useful in comparing different versions of SASAT later in this paper. Before providing the distributions, however, it is instructional to consider the variance of the results given above. Table 9 provides the standard deviation of the flips for SASAT.

V	C	Flips	Standard Dev.
100	425	31,863	88,117
200	850	396,341	795,558
300	1275	1,942,040	2,911,738
400	1700	2,269,500	3,718,480
500	2125	4,438,820	7,970,972

Table 9: Standard deviation of flips for SASAT.

⁸ Because we were unable to obtain the specific problems solved by GSAT, we can not assume that SASAT solves a superset of the problems that GSAT solves, however.

In each case the standard deviation is higher than the mean, indicating the presence of outliers in the data. In other words, we will expect that the distribution will contain a small number of problems that are much more difficult than average. In order to confirm this we graphed the distribution - showing the number of satisfied problems solved within a certain number of flips. Figures 4, 5, and 6 give the distributions for the 100, 300, and 500 variable problems.

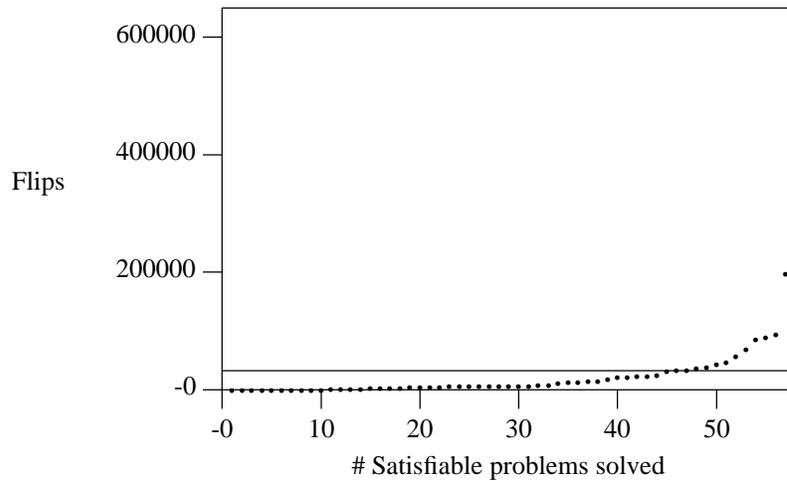


Figure 4: Distribution for SASAT on 100 variable problems.

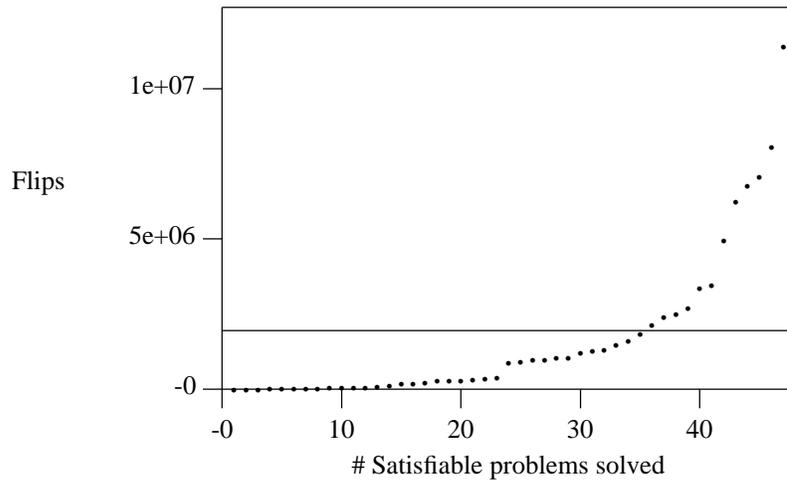


Figure 5: Distribution for SASAT on 300 variable problems.

In Figures 4, 5, and 6 we represent the mean number of flips by a solid horizontal line. As expected, the majority of the problems were solved with less than the mean number of flips, and the presence of the outliers dramatically increases both the mean and the standard deviation.

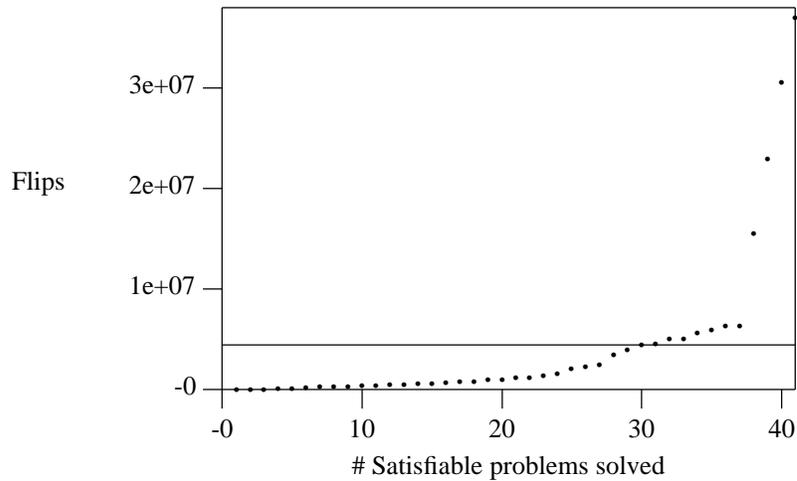


Figure 6: Distribution for SASAT on 500 variable problems.

One interesting use of these "solution" distributions is in deciding whether an unsatisfied problem is satisfiable or not. If there is any reason to believe that a problem has been drawn from a distribution similar to a Random L-SAT "problem" distribution, and SASAT has been attempting to satisfy that problem without success, Bayesian analysis can be used to estimate the probability that the problem is in fact unsatisfiable. Details of this technique can be found in Spears (1992).

4 A Modification to SASAT

As stated above, comparisons of SASAT with GSAT are difficult to make. Selman also has reported comparisons of GSAT with simulated annealing. The results have been mixed (Selman and Kautz, 1993a; Selman *et al.* 1993). Given the difficulty of making a comparison it is reasonable to wonder if SASAT is really doing better than GSAT, and if so, why? To help answer these questions we modified SASAT to make it more similar to GSAT. This is easily done by using a zero temperature logistic function that never makes a backwards move (see Figure 7). Although SASAT would still not be choosing the *best* variable to flip, it would nevertheless only make sideways or forwards moves, as GSAT primarily does.

flip variable v with probability defined by the logistic function:

```

if ( $\delta < 0$ ) return 0.0;
else if ( $\delta == 0$ ) return 0.5;
else return 1.0;

```

Figure 7. Zero temperature logistic function for SASAT.

The motivation behind this modification is the idea that backwards moves are the primary advantage of SASAT over GSAT. If this is true, we would expect the zero

temperature SASAT to perform more like GSAT, both in terms of the percentage of problems satisfied, and the amount of work required to satisfy them. In order to test this hypothesis we reran the above experiments using the zero temperature SASAT. Table 10 provides the results.

V	C	δ_s	Flips	%	i	MAX_TRIES	Time
100	425	422,000	19,853	54/100	5.4	200,000	.2 min
200	850	8,518,000	376,200	51/100	11.6	400,000	3 min
300	1275	43,578,900	1,947,180	38/100	21.0	800,000	15 min
400	1700	99,440,400	4,469,170	28/100	22.9	1,000,000	34 min
500	2125	246,889,500	11,019,800	23/100	29.3	1,600,000	83 min

Table 10: Zero temperature SASAT on hard problems.

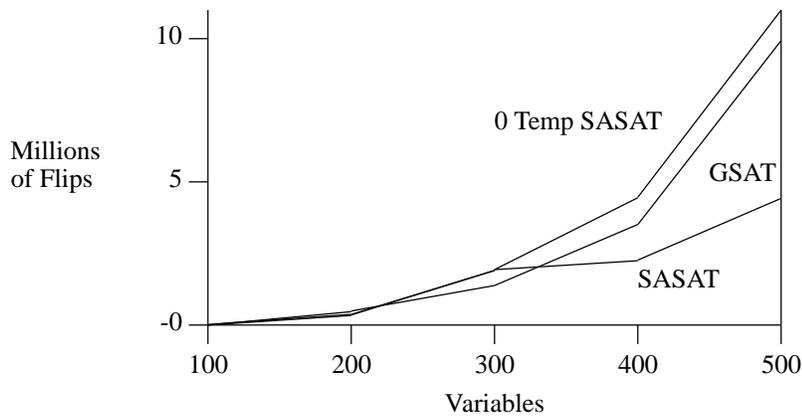


Figure 8: Comparison of GSAT, SASAT, and zero temperature SASAT.

If we consider the ratio of δ_s to flips we find that the ratio is always around 22, regardless of problem size. Zero temperature SASAT is somewhat slower than the original SASAT, in the sense that it always performs about 2200 flips per second. This speed reduction is due to the fact that zero temperature SASAT performs more δ_s per flip than the basic SASAT (22 vs 20).

Now, let us compare zero temperature SASAT with the original SASAT (and GSAT) on both the percentages and the number of flips (Figure 8). Tables 7 and 10 suggest that the percentages of problems satisfied by the zero temperature SASAT fall within the values estimated for GSAT. Furthermore, Figure 8 suggests that GSAT and zero temperature SASAT scale similarly. As expected, the zero temperature SASAT algorithm appears to behave very much like GSAT. Figures 9, 10, and 11 compare the distributions of the zero temperature SASAT with SASAT. The distribution of the modified SASAT is presented as a solid line. The distributions indicate that the relative advantage of SASAT increases as the number of variables increase.

These results highlight a number of interesting points. First, as expected, the zero temperature SASAT performs similarly to GSAT (see Tables 7 and 10, and Figure 8).

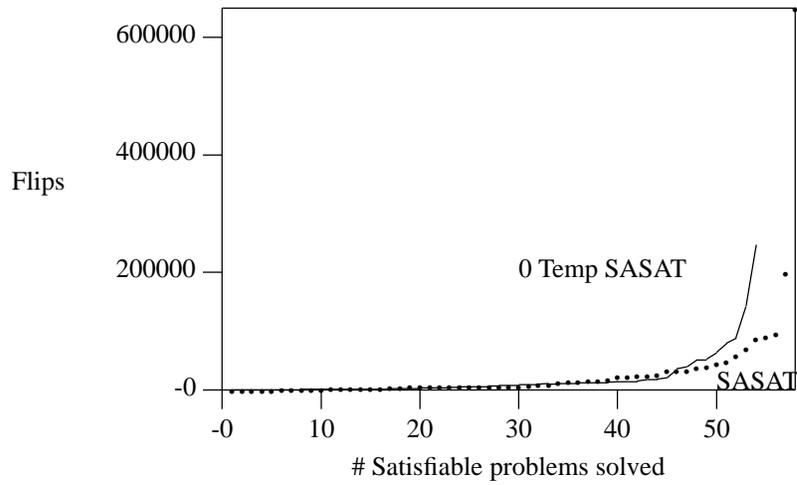


Figure 9: Distribution for zero temperature SASAT on 100 variable problems.

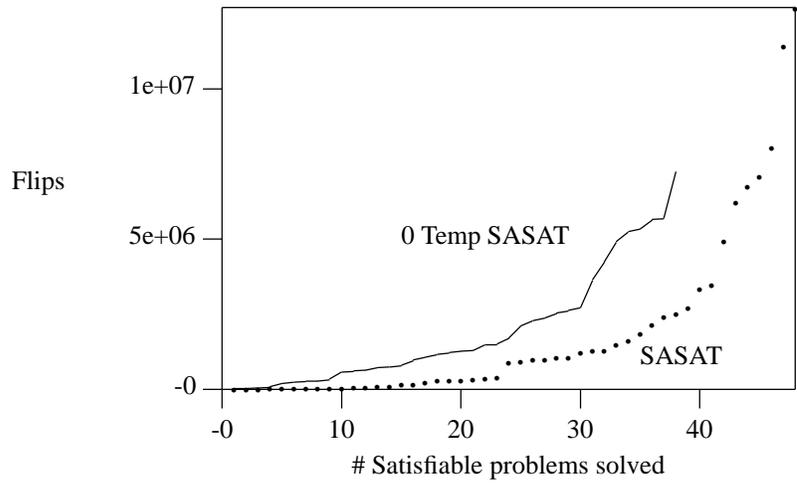


Figure 10: Distribution for zero temperature SASAT on 300 variable problems.

Second, since SASAT outperforms zero temperature SASAT, we increase our confidence in our conclusion that SASAT does indeed outperform GSAT on this class of problems, at least with the given cutoffs.⁹ Third, we have evidence to confirm that the key to the relative advantage of SASAT lies in the backwards moves (which allow SASAT to escape suboptima), since SASAT outperforms zero temperature SASAT (and GSAT) both in the percentage of problems satisfied and the number of flips required to satisfy them.

⁹ All versions of SASAT described in this paper use the same cutoffs.

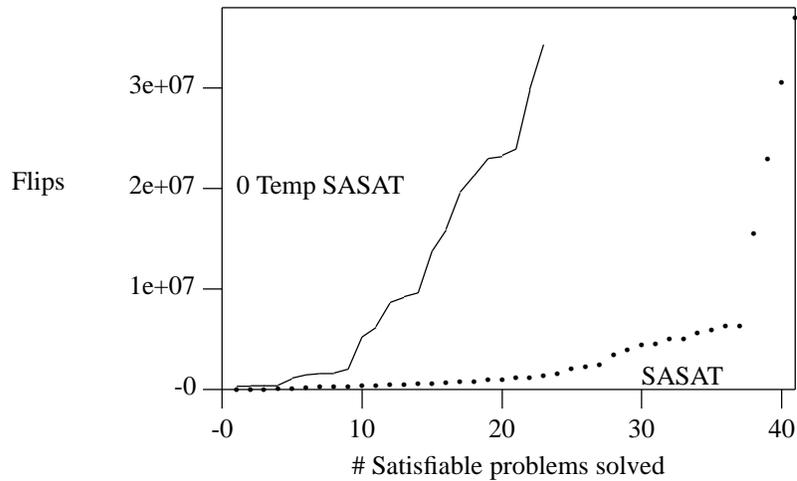


Figure 11: Distribution for zero temperature SASAT on 500 variable problems.

Despite the modification to SASAT, GSAT remains different in an important aspect. Unlike SASAT, GSAT always flips the *best* variable of the V variables that can be flipped at any time. In other words GSAT bases its decisions on the global information obtained from all variables. SASAT, on the other hand, bases its decisions only on the local information associated with one variable. Interestingly, it is not clear that the global mechanism is very useful, since zero temperature SASAT (which also uses local information) performs in a fashion that is very similar to GSAT. This raises an intriguing question. Is choosing the *best* really a good strategy? We plan to pursue this in more detail in the future.

As mentioned earlier, Selman has reported mixed results with simulated annealing. Their method is sufficiently different to make strong conclusions difficult (e.g., they always accept forward and sideways moves); however, it appears as if the annealing schedule we use can help explain some of those results. In Selman's experiments relatively high maximum temperatures were used - namely, 5 or 10. SASAT has a maximum temperature of 0.3. It is likely that temperatures much higher than 0.3 simply result in a lot of wasted search.

Finally, although it is clear that backwards moves help the performance of SASAT, it remains an open issue whether the success is due to the annealing schedule per se, or whether some simpler method of applying backwards moves would yield comparable performance (i.e., applying backwards moves with some fixed probability). As we shall see, the next section suggests that backwards moves applied according to a simple heuristic can also augment performance.

5 SASAT with a Random Walk

As mentioned earlier, one characteristic of SASAT is that it allows arbitrary sequences of backwards moves. Recently, GSAT has been enhanced by a feature referred to as a random walk (Selman *et al.*, 1993). The purpose of the walk is to allow GSAT to escape from local optima by making backwards moves. However, the random walk moves are more purposeful than those made by SASAT. Periodically, GSAT

randomly chooses an unsatisfied clause, and flips the value of a random literal within that clause (thus making that clause true). Preliminary results indicate that this is an effective heuristic for GSAT.

Considering the relative advantages that SASAT appears to have, it is reasonable to also consider adding a similar heuristic to SASAT. One elegant way is to modify the logistic function as shown in Figure 12.

flip variable v with probability defined by the logistic function:

```

with probability  $p$  {
  if ( $v$  is in an unsatisfied clause) return 1.0;
  else return 0.0;
}
else with probability  $1 - p$  {
  return  $\frac{1}{1 + e^{-\frac{\delta}{\text{temperature}}}}$  ;
}

```

Figure 12. Random walk logistic function for SASAT.

Thus, with probability p we check to see if a variable is in an unsatisfied clause. If it is we flip it. If not, we leave it alone. Finally, with probability $1 - p$ we use the standard logistic function. The motivation is to add backwards moves to SASAT that are not simply random. In fact, random walk moves are targeted towards those clauses that appear to be giving the algorithm difficulty.

Of course, the behavior of this algorithm depends greatly on the value chosen for p . Results from the genetic algorithm community suggest that such perturbations should occur roughly once for each pass over the V variables (Baeck, 1993). Drawing on these results, and some preliminary empirical experiments, we set p to $1 / V$ and reran SASAT. Table 11 presents the results.

V	C	δ_s	Flips	%	i	MAX_TRIES	Time
100	425	102,900	8,072	55/100	2.8	200,000	.05 min
200	850	1,833,400	125,239	56/100	5.0	400,000	.8 min
300	1275	19,202,100	1,236,040	53/100	11.7	800,000	8 min
400	1700	34,179,200	2,128,179	46/100	11.7	1,000,000	13 min
500	2125	99,556,000	6,016,920	46/100	15.9	1,600,000	37 min

Table 11: SASAT with random walk on hard problems.

If we consider the ratio of δ_s to flips we find that the ratio is around 16. This is because the random walk results in more flips than the original SASAT, reducing the number of δ_s computed per flip. As a result, SASAT with random walk also performs more flips per second, roughly 2700.

Again let us also compare the algorithms on the percentages of problems solved (see Tables 8 and 11), and the number of flips required to solve them (Figure 13). SASAT with the random walk solves a higher percentage of problems than it did before, achieving almost 50% on the harder problems. In terms of the average number of flips, it

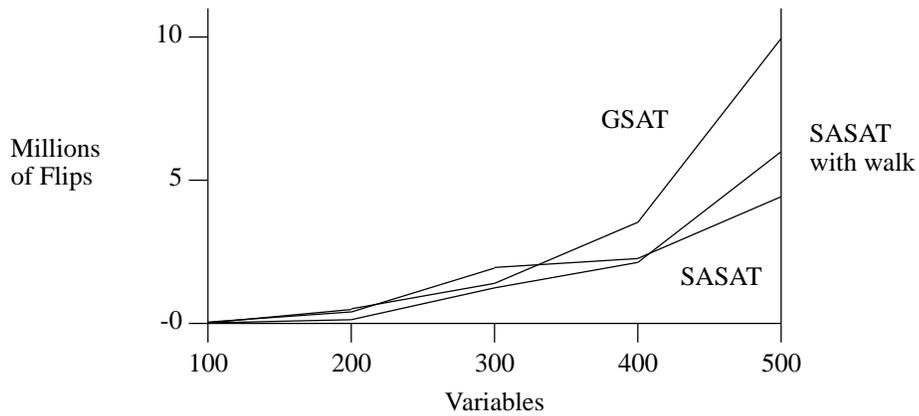


Figure 13: Comparison of GSAT, SASAT, and SASAT with random walk.

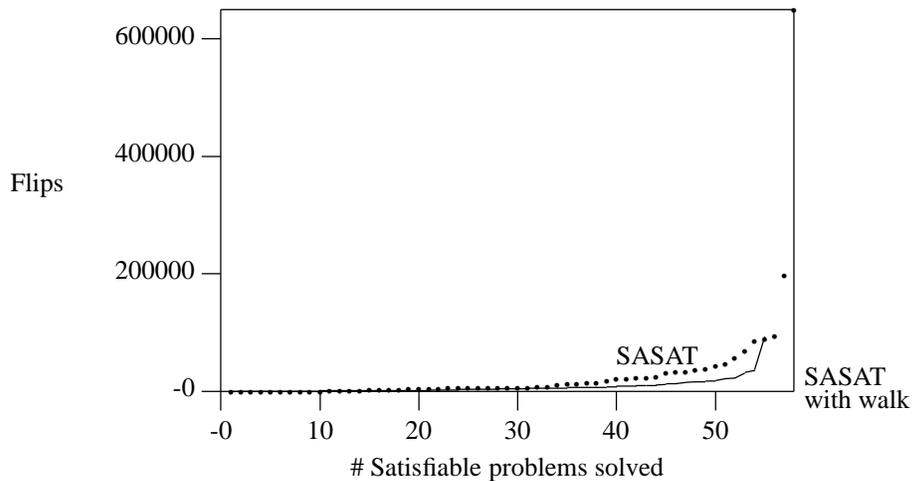


Figure 14: Distribution for SASAT with walk on 100 variable problems.

does not appear to scale as well as SASAT. However, this is somewhat misleading, since SASAT with the random walk is solving more problems. Figures 14, 15, and 16 compare the distributions of SASAT with the random walk against SASAT. The distribution for SASAT with the random walk is presented as a solid line. Note that, with the possible exception at 100 variables, SASAT with the random walk appears to be a definite improvement over the basic SASAT algorithm.

This section indicates that backwards moves applied according to a simple heuristic can also augment performance in SASAT. What is not clear is whether the performance is due to the random walk, the annealing schedule, or some combination of the two. One obvious control study would be to rerun the zero temperature SASAT with random walk. We intend to pursue this in the near future.

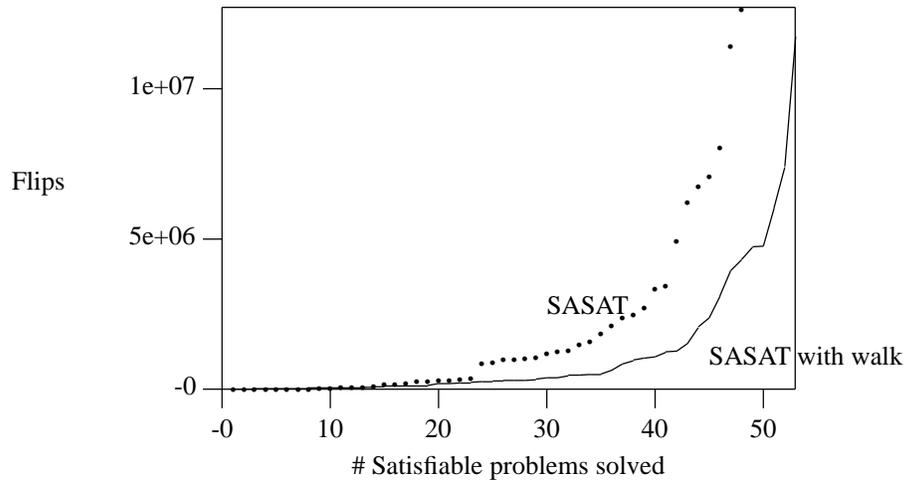


Figure 15: Distribution for SASAT with walk on 300 variable problems.

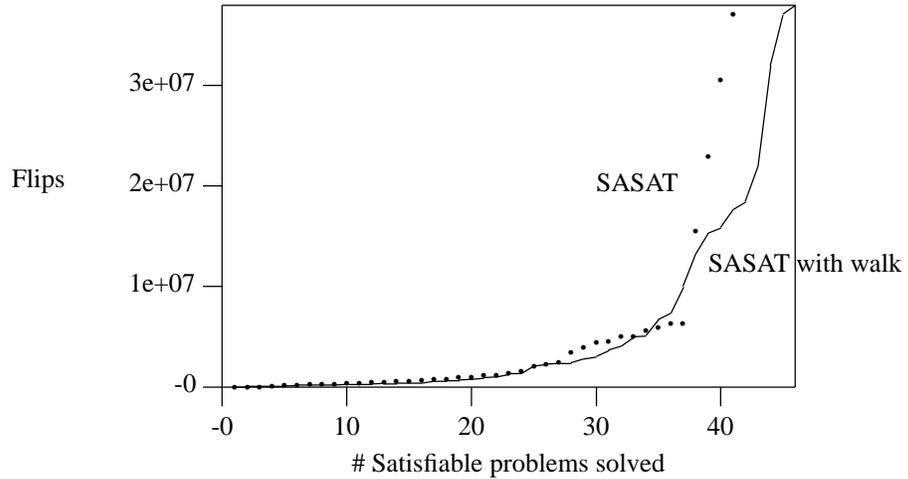


Figure 16: Distribution for SASAT with walk on 500 variable problems.

6 Scaling Issues and Harder Problems

As stated earlier, one important issue is in how to compare the performance of various algorithms. For Random 3-SAT problems where R is 4.25, GSAT computes roughly 25 δ s per flip. Fortuitously SASAT without random walk computes roughly 20 δ s per flip. Given the current implementations of SASAT and GSAT, and the problem class explored, flips appears to be a good measure for comparison.

However, suppose we consider Random L -SAT problems where L is greater than 3. Should we still compare GSAT and SASAT using flips? Recall that when L is 3 and R is 4.25, roughly 50% of the random problems are satisfiable. To maintain the 50% proportion, R must increase when L increases, which greatly increases the ratio of δ s to

flips in GSAT. Table 12 gives the values of R where roughly 50% of random problems appear to be satisfiable (Selman - personal communication), and compares the ratio of δ s to flips for both algorithms. The results for GSAT are obtained using the previously derived expression $L * R * (L - 1)$, while the results for SASAT are determined experimentally for V equal to 500.¹⁰ It is clear that in terms of the amount of δ s computed per flip performed, SASAT scales better as L increases. It is also clear that flips is not a good measure for comparison as L increases, and that both δ s and flips should be reported when L is greater than 3.

L	R	GSAT δ /flip	SASAT δ /flip	SASAT flips/sec
3	4.25	25.5	22	2500
4	9.7	116	38	590
5	21.0	420	93	82
6	43.5	1,305	163	21

Table 12: Comparison of the δ /flip ratio as L increases.

On a related note, the efficient performance of GSAT is due to the fact that it is only necessary to compute the δ s of those variables that share one or more clauses with the last flipped variable. It is also possible to make a similar improvement to SASAT, thus reducing the number of δ s computed per flip in SASAT (see the example using Tables 5 and 6 earlier in this paper). Note that, as with GSAT, this will not change the semantics of the algorithm. Experiments with SASAT on Random L -SAT problems indicate that although this is not a clear win for larger L , it does dramatically reduce the number of δ computations per flip (and increase the number of flips per second) when $L = 3$ (see Table 13).¹¹

L	R	SASAT δ /flip	Improved SASAT δ /flip	Improved flips/sec
3	4.25	22	11	3000
4	9.7	38	35	573
5	21.0	93	82	87
6	43.5	163	152	27

Table 13: Comparison of the δ /flip ratio as L increases.

A second important issue is the selection of problem instances to be solved. To avoid overfitting an algorithm to particular problems, it is important to consider large sets of problems or to consider random problems drawn from distributions. In this paper we have concentrated on a set of hard problems drawn from the Random 3-SAT distribution, ranging from 100 to 500 variables. Due to time constraints, it is difficult to produce distributions for problems with more variables. However, we can still consider harder individual instances. GSAT has solved Random 3-SAT problems of 600, 1000, and 2000 variables. Table 14 presents the results for SASAT on these problem instances.¹²

¹⁰ Again, preliminary experiments with GSAT indicate this is a reasonable estimate.

¹¹ The SASAT implementations used in Sections 3, 4, and 5 did not use this modification. However, we use it for the remainder of the paper.

¹² The notation 1.8E5 means $1.8 * 10^5$. Also, due to time constraints, for these experiments and all remaining experiments, SASAT is run once for each problem. The results for GSAT are not available.

Problem	V	C	δ_s	Flips	i	MAX_TRIES	Time
f600	600	2,550	1.8E5	15,603	1	20,000,000	.1 min
f1000	1,000	4,250	1.8E9	1.5E8	59	20,000,000	833 min
f2000	2,000	8,500	9.1E8	8.1E7	20	20,000,000	450 min

Table 14: SASAT on harder Random L-SAT problems

It could be argued that the Random L-SAT problems are not of interest, since they may not occur in realistic problems. To address this criticism, GSAT has also been used to solve larger instances, based on problems from other communities. The largest instances are based on hard graph problems. Table 15 presents the results for SASAT on these larger problems. Surprisingly, SASAT solves all four, two of them quite easily.

Problem	V	C	δ_s	Flips	i	MAX_TRIES	Time
g125.17	2,125	66,272	1.7E9	3.4E7	45	20,000,000	7563 min
g125.18	2,250	70,163	2.1E6	43,007	1	20,000,000	8 min
g250.15	3,570	233,965	3.4E5	6,193	1	20,000,000	2 min
g250.29	7,250	454,622	5.4E8	5.7E6	7	20,000,000	6667 min

Table 15: SASAT on very large problems.

As stated earlier, one motivation for the Random L-SAT problems is that they are difficult for traditional complete algorithms. Likewise, although the graph problems in Table 15 do not appear to be hard for incomplete algorithms such as GSAT and SASAT, their sheer size also makes them difficult for complete algorithms, in the sense that the search space is too large for systematic exploration.¹³ Not surprisingly, these results indicate that one potential problem class well suited to incomplete algorithms are precisely those problems that are too large for complete algorithms.

One task of the 1993 DIMACS challenge was to produce a test suite of satisfiability problems that hopefully could be useful in determining the important biases of various satisfiability algorithms, indicating the classes of problems that are well (and poorly) matched to each algorithm. Particular instances of Random 3-SAT problems and the above graph instances are included in this test suite. Also included are instances derived from parity, inductive inference, and coloring problems. The complete test suite is composed of 41 instances, 27 of which are satisfiable. Due to the size of some of these instances, we were only able to run SASAT once on each satisfiable instance. No attempt was made to optimize the parameter settings of SASAT. MAX_TEMP and MIN_TEMP were kept at 0.3 and 0.01 respectively, while MAX_TRIES was set at 20,000,000. The results are presented in Appendix 1. CPU time is also included with these results, because it is hard to compare SASAT with other satisfiability algorithms in terms of δ_s and flips (see also Appendix 2, which benchmarks the speed of the machine used). However, it should be noted that SASAT is not implemented as efficiently as possible (e.g., it appears as if we would gain a sizeable performance improvement using GSAT's data structures), and the reader should concentrate more on qualitative trends rather than on specific values.

¹³ The basic GSAT algorithm was faster than SASAT on the 2250 and 3570 variable problems, but was unable to solve the 2125 and 7250 variable problems (Selman and Kautz, 1993b).

Of the 27 satisfiable problems, 15 were solved by SASAT. SASAT performed well on the graph ("g") and inductive inference ("ii") instances, solving each one. The "ssa" and "parity" instances were much more difficult - SASAT solved only the simplest parity instances. Perhaps the biggest surprise was the difficulty of the "aim" instances - SASAT was only able to solve 2 of the 4 satisfiable instances.¹⁴

Unfortunately, what is not clear is *why* SASAT is having trouble on the "ssa," "parity," and "aim" problems. However, two intriguing observations can be made. First, SASAT is having trouble on problems where the clause to variable ratio is 4 or less. We should not draw the conclusion that *any* instance with a clause to variable ratio of less than 4 is difficult for SASAT, because this is not true of Random 3-SAT instances. However, perhaps *hard* instances are often more difficult for SASAT when the clause to variable ratio is low. Second, an analysis of the ratio of δ s to flips also shows an interesting trend. Generally, SASAT has difficulty in solving instances when this ratio is low as well. The explanation for both of these observations remains unclear and is an interesting item for future work.

7 Summary and Future Work

In this paper we consider an application of simulated annealing (SASAT) to a class of hard problems and compare the resulting algorithm with a greedy algorithm (GSAT). With the given cutoffs, SASAT appears to satisfy at least as many hard SAT problems as GSAT, with less work. We then present evidence confirming that the relative advantage of SASAT lies in its use of random backward moves, which help avoid local optima. By adding a random walk heuristic, SASAT is shown to solve even more problems. Finally, SASAT is run on a test suite of instances. The results indicate that SASAT's forte may be problems that are too large for systematic exploration by complete algorithms (e.g., see Table 15).

There are a number of potential items for future work. SASAT uses a very simple annealing schedule, which is not optimized. Recent work in adaptive annealing schedules holds the promise of improved performance. Secondly, it should be possible to add domain-dependent operators to SASAT. For example, it may be possible to add stochastic operators based on the Davis-Putnam satisfiability algorithm. Thirdly, SASAT is intrinsically parallel and is a good match for parallel architectures. Currently, Andrew Sohn of the New Jersey Institute of Technology is porting SASAT to a AP1000 multiprocessor. Fourth, the neural network version of SASAT (Spears, 1990; Spears 1993) is even more inherently parallel (since each neuron can be placed on an individual processor), and is a good match for a SIMD (Single Instruction Multiple Data) machine. Given that the neural network algorithm also makes no assumptions about the form of the boolean expression, we feel that this algorithm is very promising. Finally, there remains an open question as to precisely why certain satisfiability instances are hard or easy for incomplete algorithms such as SASAT. We intend to explore this question in the near future.

Acknowledgements

Thanks to Diana Gordon and Ken De Jong for many helpful comments on this work and this paper. Diana was also highly influential in my efforts to find a fair comparison between GSAT and SASAT. I would also like to thank David Johnson, Bart

¹⁴ SASAT with random walk solved all the "aim" instances, and another of the "ssa" instances, so it is clear that improvements are possible. However, we do not discuss these results in this paper.

Selman, Ian Gent, Toby Walsh, Antje Beringer, and John Grefenstette for provocative and insightful comments. Finally, I thank Andrew Sohn for expressing an interest in parallelizing SASAT. Any remaining errors are of course the author's responsibility.

References

Baeck, Thomas (1993) Optimal Mutation Rates in Genetic Search, *International Conference on Genetic Algorithms*, University of Illinois, Urbana - Champaign, June 1993, pgs. 2 - 8.

Crawford, J. M. & L. D. Auton (1993) Experimental Results on the Crossover Point in Satisfiability Problems, *Proceedings of the 1993 AAAI Conference*, Washington, DC, pgs. 21 - 27.

De Jong, K.A. & W. Spears (1989) Using Genetic Algorithms to Solve NP-Complete Problems, *International Conference on Genetic Algorithms*, George Mason University, Fairfax, Virginia, June 1989, pgs. 124 - 132.

Gu, J. (1992) Efficient Local Search for Very Large-Scale Satisfiability Problems, *SIGART Bulletin*, 3(1), January 1992.

Mitchell, D., Selman, B., & H. Levesque (1992) Hard and Easy Distributions of SAT Problems, *Proceedings of the 1992 AAAI Conference*, San Jose, CA, pgs 459 - 465.

Selman, B., Kautz, H. A., & B. Cohen (1993) Local Search Strategies for Satisfiability Testing, paper in preparation for the 2nd DIMACS Challenge, Rutgers University.

Selman, B., & H. A. Kautz (1993a) Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems, *Proceedings of IJCAI-93*.

Selman, B., & H. A. Kautz (1993b) An Empirical Study of Greedy Local Search for Satisfiability Testing, *Proceedings of the 1993 AAAI Conference*, Washington, DC, pgs 46 - 51.

Selman, B., Levesque, H., & D. Mitchell (1992) A New Method for Solving Hard Satisfiability Problems, *Proceedings of the 1992 AAAI Conference*, San Jose, CA, pgs 440 - 446.

Spears, W. M. (1993) A NN Algorithm for Hard Satisfiability problems, Artificial Intelligence Center Internal Report #AIC-93-014, Naval Research Laboratory, Washington, DC 20375.

Spears, W. M. (1992) Probabilistic Satisfiability, Artificial Intelligence Center Internal Report #AIC-92-026, Naval Research Laboratory, Washington, DC 20375.

Spears, W. M. (1990) Using Neural Networks and Genetic Algorithms as Heuristics for NP-Complete Problems, *Masters Thesis*, Department of Computer Science, George Mason University, Fairfax, Virginia.

Young, R. A. & A. Reel (1990) A Hybrid Genetic Algorithm for a Logic Problem", *Proceedings of the 9th European Conference on Artificial Intelligence*, pp.744-746, Editor: Aiello, L.C. Publisher: Pitman, London, UK.

Appendix 1

Name	Sat?	V	Clauses	Runs(Fail)	δ_s	Flips	Time
aim-100-2_0-no-1	No			0			
aim-100-2_0-no-2	No			0			
aim-100-2_0-no-3	No			0			
aim-100-2_0-no-4	No			0			
aim-100-2_0-yes1-1	Yes	100	200	1(1)			*
aim-100-2_0-yes1-2	Yes	100	200	1(0)	5.6E8	9.6E7	232 min
aim-100-2_0-yes1-3	Yes	100	200	1(0)	8.4E8	1.5E8	375 min
aim-100-2_0-yes1-4	Yes	100	200	1(1)			*
bf0432-007.cnf	No			0			
bf2760-001.cnf	No			0			
dubois20.cnf	No			0			
dubois21.cnf	No			0			
f400.cnf	Yes	400	1700	1(0)	2.1E6	1.9E5	1 min
f800.cnf	Yes	800	3400	1(0)	7.4E8	6.4E7	450 min
f1600.cnf	Yes	1600	6800	1(1)			
f3200.cnf	Yes	3200	13600	1(1)			
f6400.cnf	Yes	6400	27136	1(1)			
g125.17.cnf	Yes	2125	66272	1(0)	1.7E9	3.4E7	7563 min
g125.18.cnf	Yes	2250	70163	1(0)	2.1E6	43007	8 min
g250.15.cnf	Yes	3750	233965	1(0)	3.4E5	6193	2 min
g250.29.cnf	Yes	7250	454622	1(0)	5.4E8	5.7E6	6667 min
ii32b3.cnf	Yes	348	5734	1(0)	1.4E7	2.2E6	90 min
ii32c3.cnf	Yes	279	3272	1(0)	4.1E7	7.5E6	203 min
ii32d3.cnf	Yes	824	19478	1(0)	3.8E6	4.6E5	20 min
ii32e3.cnf	Yes	330	5020	1(0)	1.3E7	2.0E6	65 min
par16-2-c.cnf	Yes	349	1392	1(1)			
par16-4-c.cnf	Yes	324	1292	1(1)			
par32-2-c.cnf	Yes	1303	5206	1(1)			
par32-4-c.cnf	Yes	1333	5326	1(1)			
par8-2-c.cnf	Yes	68	270	1(0)	15228	3294	.02 min
par8-4-c.cnf	Yes	67	266	1(0)	1.7E5	44307	.2 min
pret150_25.cnf	No			0			
pret150_75.cnf	No			0			
pret60_25.cnf	No			0			
pret60_75.cnf	No			0			
ssa0432-003.cnf	No			0			
ssa2670-141.cnf	No			0			
ssa7552-038.cnf	Yes	1501	3575	1(1)			*
ssa7552-158.cnf	Yes	1363	3034	1(1)			
ssa7552-159.cnf	Yes	1363	3032	1(1)			
ssa7552-160.cnf	Yes	1391	3126	1(0)	1.4E9	3.4E8	2925 min

* = However, SASAT with random walk has solved this problem.

Table 16: SASAT on the DIMACS test suite.

Appendix 2

Type of machine: Sun Sparc 10

Compiler and Flags used: cc -O4

r100.5	r200.5	r300.5	r400.5	r500.5
0.05	1.42	12.41	76.99	298.70

Table 17: User time results