Weighting for Godot: Learning Heuristics for GSAT

Jeremy Frank *

frank@cs.ucdavis.edu

(916) 752-2149

Department of Computer Science

University of California at Davis

Davis, CA. 95616

Abstract

We investigate an improvement to GSAT which associates a weight with each clause. GSAT moves to assignments maximizing the weight of satisfied clauses and this weight is incremented when GSAT moves to an assignment in which this clause is unsatisfied. We present results showing that this algorithm and its variants outperform one of the best known modifications of GSAT to date using two metrics: number of solved problems on a single try, and minimum mean number of flips to solve a test suite of problems.

Content Areas: Constraint Satisfaction, Search

1 Introduction

Local search procedures are an alternative to complete search algorithms for solving combinatorially expensive search problems. GSAT is a local search algorithm which can often find solutions to satisfiable SAT problems in Conjunctive Normal Form (CNF) quickly [SLM92]. GSAT operates by changing a complete assignment of variables into one in which the maximum possible number of clauses are satisfied by flipping a single variable.

A feature of GSAT is that often the best move it can make leaves the number of unsatisfied clauses the same. These moves are known as "sideways" moves, and GSAT typically searches regions of sideways moves for some time before finding a move which reduces the number of unsatisfied clauses. This means GSAT must randomly search these "plateaus" until it finds a way off. Much research into GSAT has focused on reducing the impact of plateau search, typically by encouraging exploration of parts of the space that GSAT has not explored yet.

A modification of GSAT that has, until recently, received little attention is the "clause weights" scheme proposed in [SK93]. In this version, each clause has an associated weight, and these weights are modified during search in order to better inform the variable selection heuristic. The criterion for variable selection becomes "maximize the weight of satisfied clauses". Selman and Kautz report that this algorithm can handle certain "gerrymandered" graphs that GSAT had difficulty solving. Cha and Iwama [CI95] present an analysis of this algorithm in comparison to GSAT and GSAT with Random Walk on K-SAT formulae with desired characteristics. In their experiments they use a single try with a fixed value of MaxFlips. They conclude that it outperforms all variants of GSAT they tested it against. Davenport et. al. use a similar scheme in GENET applied to a connectionist architecture [DTWZ94], and this work is currently being extended to real-world opitimization problems such as Partial Constraint Satisfaction and the Travelling Salesman Problem.

We present a modification of weighted GSAT and show that our version has better performance than one of the best known variants of GSAT to date. We characterize the performance of our version with respect to increasing MaxFlips for a single try as well as analyze its best performance with an optimal number of MaxFlips. We present several variants of weighted GSAT in an attempt to improve the performance of the algorithm and also to improve our understanding of the process.

In §2 we discuss variants of weighted GSAT. In §3 we discuss the experiments we ran to gather empirical data on the performance of our variants, and our analysis is presented in §4. In §5 we discuss conclusions and future work.

2 GSAT and Variants

We give the familiar algorithm outline for GSAT in figure 1. GSAT is a local search algorithm which typically employs greedy hill-climbing [SLM92] designed to solve CNF formulae; it is used to solve K-CNF and CNF formulae [SK93]. GSAT begins with a randomly generated initial truth assignment, then hill-climbs by reversing or "flipping" the assignment of

*This research was supported by Caelum Research Corp. at NASA Ames Research Center and by NSF CCR-94-0365.

```
procedure GSAT(Σ,MaxFlips, MaxTries)
  for i=1 to MaxTries
    A = gen_assignment
    for j = 1 to MaxFlips
      if solved_problem(A)
        return A
      else PossFlips = select(Σ,A)
        V=pick(PossFlips)
        A=A with V's value flipped
      end else
    end for
  end for
end

procedure select(Σ,A)
PossFlips=ε
Best = -∞
  for i=1 to Number of Vars
    A=A with V_i's value flipped
    if eval(A, Σ) =Best
      PossFlips=PossFlips ∪ V
    else if eval(A, Σ)>Best
      PossFlips=V
      Best = eval(A, Σ)
    A=A with V's value flipped
  end for
end
```

Figure 1: GSAT Algorithm Sketch.

the variable which increases the number of satisfied clauses the most. If our CNF Σ has C clauses, then $eval = \sum_{i=1}^{C} Sat_i(A)$ where $Sat_i(A) = 0$ if clause i is unsatisfied in assignment A and 1 if it is satisfied. If PossFlips contains more than one variable, the algorithm must now pick one variable to flip. Different GSAT variants control how many and which variables are examined to determine the best flip and tie-breaking schemes to pick among equally good flips. [SK93] and [GW95] have also studied random walk as a method to handle plateaus. These methods force GSAT to examine different parts of the solution space on the plateau in order to move off of it as quickly as possible.

Selman and Kautz [SK93] describe a modification in which GSAT associates a weight with each clause. The weights of all clauses that remains unsatisfied at the end of a try are incremented. A clause that has remained unsatisfied through many tries will have a higher weight than a clause which has been satisfied in most of those tries. Each clause's weight is initialized to 1, so the first try is controlled as it would be in GSAT. GSAT then picks the flip that satisfies clauses with the highest total weight. So if W_i denotes the weights of the clauses then our function $eval$ can be written $eval = \sum_{i=1}^{C} W_i * Sat_i(A)$. This version of GSAT has proven effective at solving "gerrymandered" graphs which GSAT cannot solve [SK93]. We shall refer to this algorithm as WGSAT, not to be confused with WSAT, the random walk version of GSAT [SK93].

We saw several modifications which we felt might improve the algorithm's performance and illuminate the effect that weights have on GSAT's performance, leading to even better improvements in the future. We briefly describe these modifications below.

2.1 When to Change the Weights?

Selman and Kautz originally suggested altering the weights after each try completed. This has the effect of allowing GSAT to drive the number of satisfied clauses to a very low number relative to the initial number; as a result, few weights change after each try and feedback must wait until the end of a try. We decided to update the clause weights after each *flip* instead of after each try. Now unsatisfied clauses provide feedback to the variable selection heuristic every variable flip. This has the effect of emphasizing variables in unsatisfied clauses almost immediately, without waiting for the try to complete.

2.2 Functions of Weights

We observed that *relative magnitudes* of clause weights control variable selection. If a single clause has a high weight, it may be able to sway the heuristic to cause several clauses of lower weight to become unsatisfied, but it alone can't sway a clause of higher weight than its own. Over time, clause weights may become static relative to each other. Also, we see that an increment of 1 is significant early in search but becomes less so. Finally, clauses accumulate weight in the context of assignments which are constantly changing. One way to handle this is to use different functions of the weights to drive variable selection. We chose to sum the power of the weights, thereby giving high-weighted clauses even more ability to sway the flip decision process. So $eval$ has the form $eval = \sum_{i=1}^{C} W_i^{\alpha} * Sat_i(A)$. Notice that if α is arbitrarily high that we are forced to flip a variable causing one of the highest weighted unsatisfied clause to become satisfied.

2.3 Prior Weights

As WGSAT makes its first flips many clauses are likely to be unsatisfied, while as the try continues we expect the number of unsatisfied clauses to decrease. Hence the early clause weights are "noisy", which can result in poor initial guidance to WGSAT. While poor initial guidance can eventually be overcome, we examined ways in which this noise can be damped. Since relative weights are important in variable selection, we decided to experiment with different initial values for the weights. Suppose a clause's initial weight is set to x. Then after the first flip some clauses will have weight x and others will have weight $x + 1$ and the heuristic now requires x clauses of weight $x + 1$ to outweight a single clause of weight x. If the weight is set too high, WGSAT will not benefit from its weights for a long time, effectively nullifying the impact of the weights.

2.4 Unsatisfied Variables

The observation that arbitrarily high α results in flipping a variable in the highest weighted unsatisfied clause led us to ask: what if we only attempted to flip variables in unsatisfied clauses? This method fails in GSAT because many plateau moves result from flipping variables in only satisfied clauses. However, in WGSAT the weights may force more vigorous exploration than was possible in GSAT under these conditions. This method is also different from WSAT in that there is no randomness involved; we simply don't examine a variable unless it is in at least one unsatisfied clause. Since this procedure is in some sense different than other variants of WGSAT we refer to it as UGSAT. If we examine the **select** procedure in figure 1 we notice UGSAT only needs to try and flip those variables in unsatisfied clauses. UGSAT may make fewer "exploratory" flips than other versions, resulting in time savings not accounted for by analyzing the number of state-changing flips required to solve problems. This effect may be counterbalanced by require more flips to solve some problems due to its limited ability to explore before changing the assignment. Other efforts have focused on unsatisfied variables ,in particular employing random walk on unsatisfied variables [SK93], [GW95]. Unlike these efforts, we use only the unsatisfied variables as candidates for the next flip instead of randomly flipping one of these variables.

3 Experiments

We tested our algorithms on problems found near the phase transition for 3-CNF [CKT91]; unless otherwise stated, the algorithms were tested on problems for which the number of clauses C and the number of variables N obeyed the constraint $\frac{C}{N} = 4.3$. We created the problem instances randomly; that is, we generated each clause of each problem by selecting 3 of N literals without replacement and negating each literal with probability $\frac{1}{2}$. We generated test suites of problems which were proven to have solutions using a variant of the Davis-Putnam procedure.

We decided to run a set of tests using only a single try of our WGSAT variants. We are interested in the situation where a practitioner is attempting to solve problems with a fixed amount of time, and would like to know which algorithm is more likely to solve more problems. For this set of experiments we computed the number of problems of a test suite that each variant solved, and for those problems which were solved how much time (i.e. how many variable flips) were required. Gent and Walsh found that HSAT performed better than most other versions of GSAT they tested after finding the value of MaxFlips which minimized the average number of flips to solve a suite of test problems [GW95]. We ran tests of HSAT on problems with 100 variables with a single try with MaxFlips=10000; HSAT managed to solve only 30% of the problem instances after 3000 flips, and failed to improve significantly thereafter. We ran the same tests wigh GSAT, which performed even worse than HSAT.

Flips	Mean	Sdev
1000	0.566	0.01087
3000	0.811	0.01171
5000	0.876	0.0092
10000	0.935	0.008116

Figure 2: Error Model Data.

Gent and Walsh [GW95] present results in which they determine the optimal value of MaxFlips required to solve a suite of known satisfiable 3-CNF formulae. The optimum number of flips is taken to be the number of flips resulting in the smallest mean number of flips to solve the entire suite when the number of tries is infinite. We performed this experiment for a few of our favourite variants in order to compare results as well as HSAT, which has the best performance at optimal parameter settings of any GSAT variant we know of [GW95].

We developed an error model using empirical data taken for one of our variants. Our assumption is that most of the variants are derivatives of WGSAT and therefore an error model based on WGSAT would be informative. We determined that our test suite problem instances and the random initial point were the main sources of noise in our measurements. These results were obtained by running WGSAT on a test suite of 1000 problem instances of 100 variables each which were known to have solutions. For this experiment we used WGSAT with $\alpha = 1.0$ and a prior weight of 1. We ran 1 try of WGSAT per problem and measured the number of problems which were solved. This test was performed 10 times in order to obtain figure 2. We found the standard deviation to be as much as 1.2% for low MaxFlips, and was closer to 1% when MaxFlips increased. This first test shows a marked improvement over the data we gathered for HSAT and GSAT, indicating that WGSAT performs better for a single try.

3.1 Functions of Weights

We first decided to evaluate the effect of different exponents α for the formula $eval = \sum_{i=1}^{C} W_i^{\alpha} * Sat_i(A)$. We decided to test α between 0.5 and 3.0 incremented by intervals of 0.5. We measured the number of problems from our test suite which were solved when MaxFlips was 1000, 3000, 5000 and 10000 flips. Figure 3 shows the results of this experiment. We see that for low numbers of flips that the number of solved problems is maximized when $\alpha = 1.5$, and decreases thereafter. As we expected, performance at $\alpha = 0.5$ was

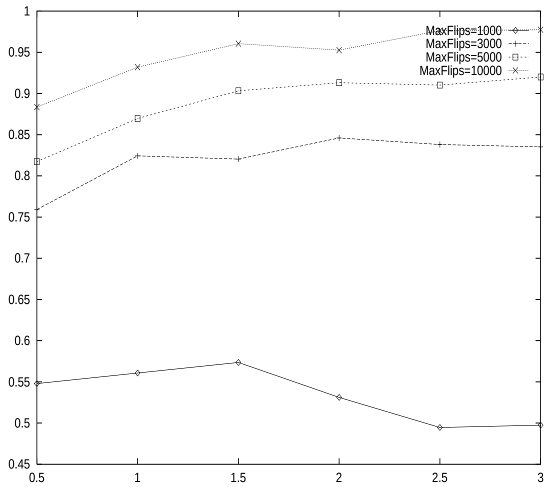

Figure 3: Satisfiability Performance of WGSAT With Varying α.

not as good as performance for higher α for MaxFlips ≥ 3000. When the number of flips increases to 3000 and beyond, we see that higher α tend to improve the proportion of problems solved, and that this effect increases as MaxFlips increased. The most marked improvement occurs between $\alpha = 0.5$ and $\alpha = 1.5$.

3.2 Priors

We decided to study the effects of different prior weights on the performance of WGSAT. We set all weights to the same initial value and tested prior weights between 1 and 20 incrementing by 5. Again, we measured the number of problems from our test suite which were solved when MaxFlips was 1000, 3000, 5000 and 10000 flips. We see in figure 4 that

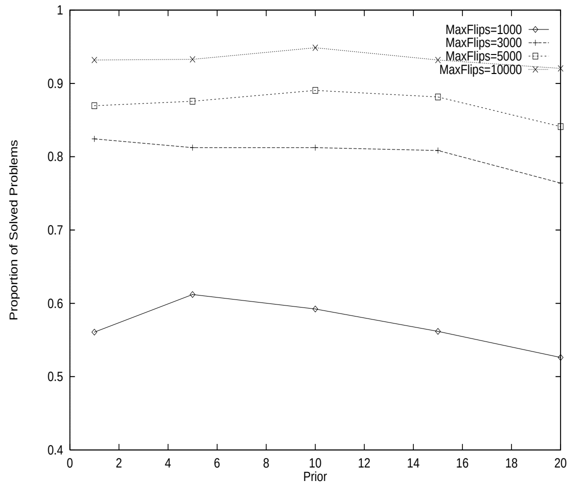

Figure 4: Satisfiability Performance of WGSAT With Prior Weights.

for low MaxFlips there is a marked peak at prior=5, while for larger MaxFlips we see that the peak moves to prior=10 (although the peaks are within 1 standard deviation according to our error model.) This indicates that there may be a shift in effectiveness of prior weights which is a function of MaxFlips. These results also indicate that noise is indeed a factor in the performance of WGSAT when MaxFlips is small, but that eventually most of the effects are overcome and we can expect only marginal improvement out of using a prior. Finally we see that with prior=20 the performance of WGSAT degrades for all MaxFlips values, indicating that WGSAT has not been able to effectively overcome this prior.

3.3 Flipping Unsatisfied Variables

We next compared the performance of WGSAT with $\alpha = 1.0$ to UGSAT. We ran UGSAT on the suite and compare the number of unsolved problems when MaxFlips was 500-10000 flips. For the sake of other comparisons, we also show in this figure the performance of WGSAT with $\alpha = 2.0$. The results of this experiment are shown in figure 5. We see that UGSAT solves fewer problems than WGSAT, but that as MaxFlips increases it appears to have similar behavior. We notice that when $\alpha = 2.0$ WGSAT performs better than when $\alpha = 1$, but the two versions of WGSAT appear to have similar asymptotic behavior in increasing MaxFlips. We also counted the to-

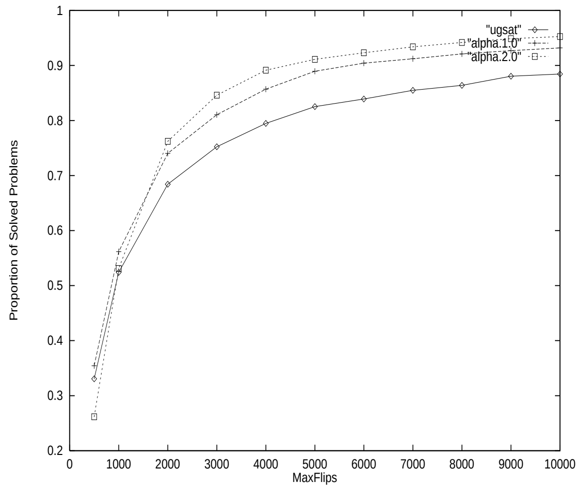

Figure 5: Satisfiability Performance of WGSAT and UGSAT.

tal number of "exploratory" flips that WGSAT and UGSAT performed; that is, the total number of flips required to determine the best flip to make. We use this measure as a surrogate for CPU time. The table in figure 6 shows the mean number of exploratory flips required to find the solutions to those problems which the algorithms solved. We see that UGSAT consistently takes between $\frac{1}{6}$ and $\frac{1}{9}$ of the time at only a slight degredation in the number of solved problems.

MaxFlips	Total Exploratory Flips		
	$\alpha = 1.0$	$\alpha = 2.0$	UGSAT
1000	44511.6	51621.9	7411.19
3000	83406	94991.9	11214.9
5000	110636	114917	13708.3
10000	137570	140218	17392.1

Figure 6: Total Number of Flips to Find Assignments.

Vars	Variant	Opt MaxFlips	Mean	SDev
100	WGSAT	1250	1591.95	2346.2
	UGSAT	1125	1932.03	2767.81
	HSAT	200	2437.27	4269.09
125	WGSAT	1750	3454.23	5159.49
	UGSAT	2000	4149.86	6261.08
	HSAT	300	5480.54	10220
150	WGSAT	2000	6982.04	12366.2
	UGSAT	2250	7902.66	11898.1
	HSAT	550	9836.46	17697

Figure 7: Optimal Performance of GSAT Variants.

These results imply that UGSAT may be a faster version of WGSAT.

3.4 Optimal Parameter Settings

We now turn to the question of performance under optimal parameter settings. In this experiment, we estimate the best performance of HSAT, WGSAT and UGSAT. We allowed each algorithm an infinite number of tries at different values of MaxFlips and counted the mean number of flips to solve the entire suite of 1000 instances; in other words we summed the number of flips required to solve all the problem instances and then divided by 1000. We found the value of MaxFlips minimizing this value. We then compared the algorithms to see which solved the test suite with the smallest mean time. We repeated this experiment for problems of 125 and 150 variables as well. The data from these experiments is shown in figure 7. We see that WGSAT with $\alpha = 1.0$ and UGSAT both outperform HSAT consistently. Not only do they require fewer flips to solve the suite, they also have a smaller standard deviation, implying that they perform more consistently than HSAT. HSAT requires many tries to solve problems, while UGSAT and WGSAT solve almost all the problems in 1 or 2 tries for 100 variable problems; as the problem size increases to 150 variables, WGSAT and UGSAT now require 3-4 tries to solve the problems. We should also note that for the larger problems WGSAT and UGSAT experienced good performance for a wide range of MaxFlips.

We can now analyze UGSAT's performance in total number of flips to solve the test suite at the optimal parameter settings. This data is presented in figure 8. The speedup column represents the ratio of WGSAT's total flips to UGSAT's total flips, thereby assessing how much faster UGSAT is than WGSAT. UGSAT outperforms WGSAT with $\alpha = 1.0$ by a factor of 5-6 consistently as the number of variables increases.

	Total Exploratory Flips		
Vars	UGSAT	WGSAT	Speedup
100	30578.2	169951	5.55
125	64479.4	441551	6.847
150	160734	1047310	6.51

Figure 8: Scaling of UGSAT Performance.

Variant	Opt MaxFlips	Mean	SDev
UGSAT	1000	1679.5	2455.31
WGSAT	1150	1447.07	2200.18

Figure 9: Optimal Performance of GSAT Hybrid Variants.

3.5 Hybrids

The above results suggest combining improvements depending on which performance measure we would like to optimize. For instance, we see that we achieve the best single try performance with WGSAT for high α and a modest prior, say 10-15. However, knowing that the best mean time to solve the test suite for WGSAT occurs at around 1000 flips for problems of size 100, we want to choose variants maximizing the number of problems solved at this value of MaxFlips. We analyzed two possible hybrids with good optimal performance potential for problems with 100 variables: UGSAT with $\alpha = 2.0$ and a prior of 5, and WGSAT with $\alpha = 2.0$ and a prior of 5. We then found the optimal value of MaxFlips for both of these variants and compared the performance of these variants to our earlier results. These comparisons are shown in figure 9. We notice that we have achieved a slight improvement in comparison to the data in figure 7, which indicates that these modifications can result in an improved GSAT algorithm.

4 Analysis

We observe that WGSAT and UGSAT continuously change the clause weights which drive the search; in effect, creating a new heuristic at every step. This is substantively different from earlier attempts to modify GSAT search. The clause weights can be interpreted as a measurement of the difficulty in satisfying the clauses which has beem "marginalized" over all assignments the search examined. This is not a true marginalization because local search (hopefully) doesn't examine every state, and thus the weights are not an objective measurement of difficulty. Also, since there may be many solutions to a particular problem instance, we assume the weights are only relevant to WGSAT's trajectory towards a particular solution or family of solutions.

The prior weights had a large impact on the performance of WGSAT only for smaller values of MaxFlips, with a much diminished impact as MaxFlips grew larger. We conclude that for a single try a prior is not important, but it is important for optimal performance. We also observe that the exponentiation schemes designed to emphasize any relative differences in weights result in better performance of WGSAT at higher values of MaxFlips. We interpret this to mean that, as more and more flips are required to solve a problem instance, the increase in weights becomes less and less meaningful to WGSAT. However, emphasizing the weights too early results in noisy behavior and poor performance. This leads us to consider schemes where the exponent WGSAT uses in the heuristic increases as a function of the flip number.

Finally, we saw that UGSAT performs comparably to WGSAT for the problems we examined but may take less time due to it's more limited exploration of possible flips. Our conclusion is that if UGSAT fails to examine a good flip early in a try it will be forced to examine similar flips later due to the feedback of the clause weights.

5 Conclusions and Future Work

We have modified Selman and Kautz's WGSAT algorithm to incorporate more general weight modification schems, prior weights and alternative heuristic functions on clause weights. Our experiments show that WGSAT is superior to GSAT and HSAT for single try performance and performance with optimal MaxFlips, and that the performance scales with increased problem size, and have suggested some improvements based on our results. We showed that clause weights contain some long-term information about how difficult GSAT finds clauses to satisfy, and that this information is apparently more important than short-term information.

We have considered extending weights to sets of clauses, variables and sets of variables in problem instances. For example, in coloring problems, sets of satisfiability clauses represent edge constraints in the original graph, and similar structures occur in other problems. We envision that weighting variables might provide information similar to that of clause weights. We have given some thought to the idea of attempting to set the prior more intelligently than merely assigning a uniform value. Since the weights act as the guide to the heuristic, setting these correctly early may yield dramatic improvement to WGSAT. Finally, recent research has focused on more structured test problems than the random problems. Examples of such problems are chain CNFs [DR94] and SAT instances drawn from crossword puzzle construction [Kon94] and existence of Quasigroups [GW95]. We would like to extend our testing to these domains, where we see some interesting problems for WGSAT. We have made some preliminary experiments using WGSAT to try and solve a satisfiability encoding of the Towers of Hanoi problem. WGSAT with $\alpha = 1$ performed very poorly on the Towers of Hanoi problem, solving it 10% of the time given MaxFlips=5000 and 1 try, while backtracking finished the problem in about 30 seconds. We would like to continue investigating these types of problems.

6 Acknowledgements

I would like to thank Peter Cheeseman, John Stutz and John Allen of NASA Ames Research Center for their input during this research, and also Ian Gent of the University of Strathclyde and Toby Walsh of IRST for their comments.

References

[CI95] B. Cha and K. Iwama. Performance tests of local search algorithms using new types of random cnf formulas. *14th International Joint Conference on Artificial Intelligence*, pages 304–310, 1995.

[CKT91] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the *really* hard problems are. *IJCAI*, pages 163–169, 1991.

[DR94] R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure revisited. *4th International Conference on Knowledge Representation and Reasoning*, 145:134, 1994.

[DTWZ94] A. Davenport, E. Tsang, C. J. Wang, and K. Zhu. Genet: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 325–330, 1994.

[GW95] I. Gent and T. Walsh. Unsatisfied variables in local search. In J. Hallam, editor, *Hybrid Problems, Hybrid Solutions*. IOS Press, 1995.

[Kon94] K. Konolige. Easy to be hard: Difficult problems for greedy algorithms. *4th International Conference on Knowledge Representation*, pages 374–378, 1994.

[SK93] B. Selman and H. Kautz. Domain independent versions of GSAT solving large structured satisfiability problems. *IJCAI*, 1993.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. *AAAI*, pages 440–446, 1992.