# Variable-Selection Heuristics in Local Search for SAT

Alex S. Fukunaga*
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, MS 525-3660
Pasadena, CA 91109-8099
alex.fukunaga@jpl.nasa.gov

July 1, 1997

## Abstract

One of the important components of a local search strategy for satisfiability testing is the variable selection heuristic, which determines the next variable to be flipped. In a greedy local search such as GSAT, the major decision in variable selection is the strategy for breaking ties between variables that offer the same improvement in the number of unsatisfied clauses. In this paper, we analyze a number of tie-breaking strategies for GSAT and evaluate the strategies empirically using randomly generated 3-SAT instances from a hard distribution of random instances. We find that the property of fairness, which was proposed in the literature as being the critical property of a successful variable strategy, is not a sufficient property, and show that randomness plays a significant role in the success of variable selection heuristics.

## 1 Introduction

Local search algorithms for propositional satisfiability such as GSAT [10] have received much attention in recent years because of the discovery that it is possible to find solutions to difficult problems which are much larger than those which are solvable with conventional, systematic approaches such as the Davis-Putnam procedure [3], at the cost of completeness.

The basic schema for a local search algorithm for satisfiability testing is the following: Initially, a complete assignment of truth values to variables is generated randomly. Then, the truth values of the variables are repeatedly flipped in order to find a satisfying solution, usually applying some hill-climbing technique that tries to minimize the number of clauses that are left unsatisfied. Since hill-climbing approaches are susceptible to getting stuck at local minima, various heuristics for escaping local minima are applied to resume progress once the hill-climbing search becomes stuck.

The best-known local search algorithm for satisfiability testing is GSAT [10], shown in Figure 1. A greedy hill-climbing procedure is repeatedly applied to a randomly generated initial assignment. In order to escape local minima, after each MAXFLIPS flips, a new random assignment is generated, and the greedy-hill climbing procedure begins again. This is repeated MAXTRIES times (or until time runs out).

A distinctive feature of GSAT is the particular method by which a variable is chosen to be flipped. In the original formulation of GSAT, Selman et al. specified that "... the variable whose assignment is to be changed is chosen at random from those that would give an equally good improvement." Their justification for this choice was that "such non-determinism makes it very unlikely that the algorithm makes the same sequence of changes over and over." [10]. However no empirical support of this conjecture was presented.

Gent and Walsh [4, 6] later questioned the importance of randomness in the method of picking the variable to be flipped, and presented experimental results using 50-100 variable random problems, as well as $N$-queeens problem instances that showed that randomness is not essential in the variable picking procedure. In addition, Gent and Walsh concluded that the property of *fairness* in variable picking is important. (According to [6], a procedure is *fair* if "..it eventually picks any variable that is offered continually").

This work was originally motivated by our experience when we implemented a GSAT engine. When testing our code using some standard benchmarks, we were surprised to find that our implementation was obtaining consistently worse results than those reported in the literature, and we discovered that the only significant difference between our implementation and that of others was our use of a fair but non-random variable selection strategy.[1]



[1]This was the FIFO strategy, described in Section 2.

**GSAT**

**Input:** set of clauses $\alpha$, MAXFLIPS, and MAXTRIES
**Output:** a satisfying truth assignment of $\alpha$, if found
**for** $i$ := 1 to MAXTRIES
  $T$ := a randomly generated truth assignment
  **for** $j$ := 1 to MAXFLIPS
    **if** $T$ satisfies $\alpha$ **then return** $T$
    $p$ := **Choose** a propositional variable to flip that maximizes the increase in total number of clauses of $\alpha$ satisfied by $T$.
    $T$ := $T$ with the truth assignment of $p$ reversed
  **end for**
**end for**
**return** *failure* (no satisfying assignment found)
**end**

Figure 1: The GSAT Procedure.

In this paper, we revisit the variable selection method used in GSAT and experimentally evaluate a number of alternative heuristics for picking variables. We show that randomness plays a much more significant role than indicated by the conclusion of [4, 6]. The rest of the paper is organized as follows. In Section 2 we present several variable picking heuristics for GSAT. An empirical comparison of these methods is presented in Section 3, and Section 4 presents an analysis of the scaling behaviors of these empirical results. Section 5 concludes with a discussion of our results.

## 2 Variable Selection Heuristics

A *variable selection heuristic* is the method used to choose the variable to be flipped at every iteration of a local search algorithm for satisfiability. In this paper, we restrict our attention to the class of algorithms based on greedy hill-climbing such as GSAT (Figure 1), which chooses a variable that leads to the greatest improvement in the number of unsatisfied clauses.[2] In this framework, the main remaining design issue is the heuristic used to break ties among variables that yield the same improvement when flipped.

A naive implementation of GSAT would, at every iteration, compute the change in the number of clauses satisfied if each variable were flipped. Since this would iterate through each variable and each clause, the complexity of each iteration of such an implementation would be $O(NM)$, where $N$ is the number of variables, and $M$ is the number of clauses. However, efficient implementations of GSAT use data structures that incrementally update this information, resulting in significant speedup. Since the complexity of various tie-breaking heuristics depends on the implementation of these data structures, we describe our incremental updating mechanism below.[3]

We define the *gain* of a variable $v$ to be the *net decrease* in the number of unsatisfied clauses if $v$ were to be flipped. A *gain bucket* is a set of variables that have the same gain. A *gain table* is an array of gain buckets, sorted in descending order of their gains. That is, the gain table groups variables according to their gains.

We now outline the incremental update procedure used when a variable is flipped. We say that a variable $v$ is *critical* with respect to clause $c$ if flipping $v$ would result in either $c$ changing its state from *satisfied* to *unsatisfied*, or from *unsatisfied* to *satisfied*. For each variable $v$, a list of all clauses $C_v$ that contain the variable is stored. When $v$ is flipped, we iterate through each clause in $C_v$. For every variable $v'$ in a clause $c \in C_v$, we incrementally update the gain of $v'$ (i.e., possibly move the variable to another gain bucket), if the criticality of $v'$ changes. For example, assume we have a clause $c = (v_0 \vee v_1 \vee v_2)$, and that the current assignment of the variables is $v_0 = true$, $v_1 = false$, and $v_2 = false$. At this point, flipping $v_1$ or $v_2$ will not affect whether the clause is satisfied or not; only $v_0$ is critical with respect to $c$. Now, suppose $v_0$ is flipped, so that $v_0 = false$. Then, $v_0$, $v_1$ and $v_2$ all become critical variables, and the gains of all three variables are increased (since flipping any one of them next would satisfy the clause). This procedure is repeated for all $c \in C_v$.

This updating procedure takes $O(C_{pv}V_{pc})$ time, where $C_{pv}$ (clauses-per-variable) is the average number of clauses that a variable appears in, and $V_{pc}$ (variables-per-clause) is the average number of variables in a clause. Normally, in experiments using randomly generated formulas, $C_{pv}$ is roughly constant (determined by the critical parameter, clauses to variable ratio [7]), and $V_{pc}$ is a fixed number (most commonly, $V_{pc} = 3$). In practice, this is several orders of magnitude more efficient than the $O(NM)$ naive update procedure, since $C_{pv}$ and $V_{pc}$ are several orders of magnitude smaller than $M$ and $N$, respectively.

Choosing a variable to be flipped is done by picking the gain bucket corresponding to the highest, non-empty gain table entry[4], then choosing a particular variable out of this bucket using one of the tie-breaking strategies described below.

[2] If it is not possible to improve the number of unsatisfied clauses, then a *sideways move* that keeps the number of unsatisfied clauses constant is chosen.

[3] The terminology used in this paper (i.e., gain, gain buckets, gain tables) is adapted from the established terminology for analogous data structures used in local search algorithms for a similar problem domain, hypergraph partitioning [1].

[4] An index to the highest non-empty gain table is maintained incrementally, so finding this entry is a constant time operation.

## 2.1 Tie-Breaking Heuristics

We now describe the tie-breaking heuristics (i.e., strategy for selecting a variable out of a gain bucket) that were evaluated. Each of the strategies are evaluated according to their complexity and their *fairness*. Gent and Walsh [6], proposed that a strategy is *fair* "...if it is guaranteed to eventually pick any variable that is offered continually." They note that this is a weak definition, since it allows a variable not to be picked if it presented, say, only every other time. With respect to the terminiology used in this paper, we define fairness as follows:

**Definition 1** *A variable selection heuristic $H$ is* fair *if the following is true: Given a variable $v$ which is inserted into a gain bucket $b$, $H$ guarantees that if $b$ is continually selected as the highest gain bucket and $v$ remains in $b$, then $v$ will eventually be selected (with probability approaching 1).*

The *Random* heuristic uses a gain bucket that is implemented as an array in which access to any element takes constant time, and the number of elements in the array is maintained. At each iteration, an element of the array is picked randomly and chosen to be flipped. This element is then deleted, and resulting "hole" in the array is filled in by moving the last nonempty element of the array into the hole. New elements are inserted into the first empty element of the array. Thus, insertions and removals are done in constant time. This is a fair strategy, since the probability of eventually picking a variable approaches 1 if the bucket is randomly sampled infinitely many times.

The *First-In-First-Out (FIFO)* strategy uses a linked list gain bucket implementation. Variables are inserted at the tail of the list, and are chosen (removed) from the head of the list, so both insertion and removal are constant time operations. FIFO is clearly fair, since a variable that is inserted into a gain bucket with $N$ variables is guaranteed to be picked after the gain bucket is chosen at most $N + 1$ times by GSAT.

The *Last-In-First-Out (LIFO)* heuristic uses a linked list gain bucket implementation. Variables are inserted and removed from the head of the list. Both insertion and removal take constant time. Intuitively, this is a poor strategy, since there is too much locality in the choice of variable – it is possible to cycle among a small subset of variables at the head of the gain bucket list, while other variables are continually "stuck" near the tail of the list and never chosen. LIFO is not a fair strategy, because even if a variable $v$ is offered continually, it is possible to never pick $v$ if at every step a new element is added to the gain bucket.

# 3 Experimental Results

The tie-breaking heuristics described above (Random, FIFO, LIFO) were evaluated experimentally using formulas generated randomly using the fixed clause length model [7]. Three parameters are controlled: the number of variables $N$, the number of literals per clause $k$, and the number of clauses $M$. For a given $N$ and $M$, a random problem instance is created by generating $M$ clauses of $k$ variables, where each clause contains $k$ distinct variables from $N$ which are negated with probability 0.5. 500 instances of 3-SAT ($k = 3$ variables per clause) problem instances were generated, where the $R = M/N$, the ratio of the number of clauses to the number of variables was fixed at 4.3. As shown in [7], this generates a distribution of instances that are difficult to solve. Problems of up to 500 variables were used.

| Parameters | | | Number Solved | | |
|---|---|---|---|---|---|
| Vars | MAXFLIPS | MAXTRIES | Random | FIFO | LIFO |
| 50 | 250 | 10 | 246 | 232 | 23 |
| 100 | 500 | 50 | 174 | 138 | 0 |
| 150 | 1500 | 100 | 170 | 84 | 0 |
| 200 | 2000 | 250 | 144 | 58 | 0 |
| 250 | 2500 | 250 | 130 | 37 | 0 |
| 300 | 6000 | 250 | 152 | 12 | 0 |
| 400 | 8000 | 450 | 83 | 2 | 0 |
| 500 | 10000 | 1000 | 18 | 1 | 0 |

Table 1: Performance of the Random, FIFO, and LIFO variable selection heuristics on difficult random 3-SAT instances. The number of instances (out of 500) that were solved by each strategy given a computational resource bound of MAXTRIES and MAXFLIPS is shown above.

For each randomly generated problem, GSAT was run using each of the tie-breaking strategies, where each run consisted of up to MAXTRIES iterations of MAXFLIPS flips (the solution was completely randomized after every MAXFLIPS flips).

Table 1 shows the results of this experiment. The number of instances for which satisfying assignments were found using each tie-breaking strategy is shown. Note that not all of the problems in the randomly generated set are satisfiable; according to previous studies [7, 2], approximately half of the 500 instances are expected to be satisfiable.[5]

As predicted, the LIFO strategy performed very poorly. Although the FIFO strategy was competitive with the randomized strategies for the smaller problems (50 and 100 variables), it performed relatively poorly on the larger problem instances. Overall, the Random strategy performed significantly better than the two non-random strategies.

Next, we tested the tie-breaking heuristics in the context of a slightly different local search, GSAT with random walk [8]. This is a simple extension of GSAT which works as follows:

- With probability $p$, pick a variable occuring in some unsatisfied clause and flip its truth assignment.
- With probability $1-p$, follow the standard GSAT scheme,

[5] It would be possible to run a efficient systematic procedure such as Tableau [2] to determine exactly how many of the smaller instances are actually satisfiable; however, this was not currently feasible for a large set of the largest problem instances.

i.e., pick randomly from the list of variables that gives the largest decrease in the total number of unsatisfied clauses.

GSAT with random walk has been shown to be significantly more effective than GSAT alone in a number of problem domains [8].

Table 2 shows the results of this experiment (using the same set of random instances as for the previous experiment). The number of instances that were solved using each tie-breaking strategy is shown. The probability of a random walk being taken, $p$, was set to 0.5.

Overall, the performance of GSAT with random walk was significantly better than without random walk. Interestingly, the performance of the FIFO and LIFO strategies were significantly enhanced in the context of random walk. In fact, FIFO seems to be the best performer overall. Although the performance of LIFO is still significantly worse than the other strategies, its performance is vastly improved compared to the LIFO without random walk.

| Parameters | | | Number Solved | | |
|---|---|---|---|---|---|
| Vars | MAXFLIPS | MAXTRIES | Random | FIFO | LIFO |
| 50 | 250 | 10 | 324 | 331 | 303 |
| 100 | 500 | 50 | 253 | 246 | 225 |
| 150 | 1500 | 100 | 245 | 251 | 227 |
| 200 | 2000 | 250 | 224 | 235 | 200 |
| 250 | 2500 | 250 | 243 | 267 | 201 |
| 300 | 6000 | 250 | 244 | 256 | 210 |
| 400 | 8000 | 450 | 199 | 235 | 145 |
| 500 | 10000 | 1000 | 78 | 103 | 40 |

Table 2: Performance of the Random, FIFO, and LIFO variable selection heuristics on difficult random 3-SAT instances, when combined with the random walk strategy of Selman and Kautz. The number of instances (out of 500) that were solved by each strategy given a computational resource bound of MAXTRIES and MAXFLIPS is shown above.

For comparison, we implemented a version of Walksat, [9], which has recently been shown to be a promising alternative to GSAT on a number of problem domains. The basic iteration of Walksat is implemented as follows:

- Randomly pick a clause that is not satisfied by the current assignment.
- Pick (using a greedy heuristic with probability $p$, or at random with probability $1-p$) a variable within that clause to flip.

Table 3 shows the result of running Walksat on the same set of random instances as for our previous two experiments with GSAT. Following previous researchers [9], the probability of using a greedy heuristic to choose a variable once a clause had been chosen was set to 0.5. Overall, the data shows that the performance of GSAT with random walk with a good choice of tie-breaking strategy is competitive with that of Walksat for this class of random formulas.

| Parameters | | | Number Solved |
|---|---|---|---|
| Vars | MAXFLIPS | MAXTRIES | Walksat |
| 50 | 250 | 10 | 281 |
| 100 | 500 | 50 | 211 |
| 150 | 1500 | 100 | 202 |
| 200 | 2000 | 250 | 190 |
| 250 | 2500 | 250 | 218 |
| 300 | 6000 | 250 | 255 |
| 400 | 8000 | 450 | 209 |
| 500 | 10000 | 1000 | 92 |

Table 3: Performance of the Walksat algorithm on our set of hard 3-SAT random instances The number of instances (out of 500) that were solved by each strategy given a computational resource bound of MAXTRIES and MAXFLIPS is shown above.

Finally, we evaluated some *hybrid selection strategies*, in which randomness was added to the non-random FIFO/LIFO strategies. The *FIFO-Random (FR) Hybrid variable selection strategy* works as follows:

- With probability $p$, select a random variable from the best gain bucket.
- With probability $1-p$, select a variable from the best gain bucket using the FIFO strategy.

Likewise, the *LIFO-Random (LR) Hybrid strategy* applies a random selection heuristic with probability $p$ and a LIFO strategy with probability $p$. These hybrid strategies were evaluated with values of $p$ ranging from 0.1 to 0.75, in the context of the standard GSAT algorithm (i.e., without random walk). Table 4 summarizes the results of this study. As expected, the performance of the hybrid strategies is similar to that of their corresponding non-random strategies (FIFO, LIFO) at low values of $p$, and performance improves as $p$ is increased, and becomes competitive with the Random selection strategy.

# 4 Analysis of the Scaling Behavior

An interesting observation from the experimental results above is that the difference in the performances of the variable selection heuristics scales nonlinearly. That is, the relative differences in the performances increases as the problem size increases (rather than remaining a constant factor difference). To understand this phenomenon, we analyze the scaling behavior of the movement of variables between gain buckets in the gain table when a single variable is flipped.

As described in Section 2, flipping a single variable $v$ results in the potential update of the gain values of each variable in every clause in each clause containing $v$. Thus, the flip of a single variable could possibly result in relatively large changes in the structure of the gain table, since all variables whose net gain is non-zero are moved from their current gain bucket to a new gain bucket.

| Parameters | | | Number Solved | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | FIFO-Random Hybrid | | | | LIFO-Random Hybrid | | | |
| Vars | MAXFLIPS | MAXTRIES | FR-0.1 | FR-0.25 | FR-0.5 | FR-0.75 | LR-0.1 | LR-0.25 | LR-0.5 | LR-0.75 |
| 50 | 250 | 10 | 246 | 251 | 255 | 251 | 146 | 196 | 244 | 243 |
| 100 | 500 | 50 | 169 | 167 | 189 | 182 | 36 | 110 | 156 | 181 |
| 150 | 1500 | 100 | 136 | 156 | 166 | 170 | 20 | 91 | 156 | 172 |
| 200 | 2000 | 250 | 112 | 126 | 145 | 153 | 9 | 66 | 132 | 150 |
| 250 | 2500 | 250 | 85 | 102 | 122 | 136 | 1 | 35 | 101 | 130 |
| 300 | 6000 | 250 | 82 | 104 | 134 | 159 | 1 | 47 | 132 | 160 |
| 400 | 8000 | 450 | 32 | 46 | 70 | 86 | 0 | 17 | 63 | 94 |
| 500 | 10000 | 1000 | 4 | 9 | 14 | 15 | 0 | 2 | 10 | 20 |

Table 4: Performance of the FIFO-Random and LIFO-Random hybrid variable selection strategies, for $p \in \{0.1, , 0.25, 0.5, 0.75\}$ on difficult random 3-SAT instances. (In the table FR-0.1 denotes the FIFO-Random hybrid with $p = 0.1$, LR-0.5 denotes the LIFO-Random hybrid with $p = 0.5$, and so on.) The number of instances (out of 500) that were solved by each strategy given a computational resource bound of MAXTRIES and MAXFLIPS is shown above.

These *side effect* movements of the variables among gain buckets can significantly disrupt the FIFO/LIFO orderings in the gain buckets.

For example, suppose that we are using a FIFO selection strategy, and a gain bucket $g$ contains variables $v_1, v_2, ...v_n$ (where $v_1$ was inserted into $g$ first, $v_2$ was inserted second, and so on). Suppose $g$ is repeatedly offered as the bucket with the highest gain. If there were no side effects, then the variables would be selected in the order inserted $(v_1, v_2, ...v_n)$. However, when side effects are present, then with every flip, it is possible that some variables are moved from $g$, and/or some new variables are added to $g$. Obviously, the lower the average number of side effects per move, the more likely it is that $g$ is undisturbed by a side effect and that the variables are selected in the original (FIFO) order.

Note that if the number of side effects is large with respect to the number of variables, then the variable selection heuristic becomes relatively unimportant – the fixed orderings imposed by the FIFO/LIFO strategies are irrelevant when a large fraction of the variables are moving in and out of the gain buckets at every flip due to side effects. The converse of this observation is that the fixed orderings (FIFO/LIFO) play a more significant role when the average fraction of the variables moving due to side effects is low.

It can be shown that on average, the maximum number of variables that move between gain buckets when a variable is flipped is independent of the size of the problem (the number of variables), and is relatively small for large problems. More formally:

**Claim 1** *For the class of random $k$-SAT instances generated by the fixed clause length model, the average number of variables moving between gain buckets is less than $k(k-1)R + 1$.*

*Proof:*

Consider the class of random $k$-SAT instances generated using the fixed clause length model (defined in Section 3). Let $v$ be a variable that is flipped. Let $C_v$ be the set of clauses that contain $v$. Each clause has $k-1$ variables other than $v$, so clearly, the number of variables that move is bounded by $(k-1)|C_v| + 1$.

The expected value of $|C_v|$ is $M \times (k \times 1/N) = k \times M/N = kR$, so the average number of variables that move between gain buckets is less than $k(k-1)R + 1$.□

This bound of $k(k-1)R + 1$ is independent of $N$. As the problem size grows (i.e., $N$ increases), the fraction of the variables that move between gain buckets on each flip (which is bounded by $(k(k-1)R + 1)/N$) decreases. For small $N$, each flip moves a relatively large fraction of the variables between gain buckets, compared to large $N$. This partially explains why the performance differences between the variable selection heuristics depends on $N$: The smaller $N$ is, the more disruptive each flip is to the gain table structure, and hence, the less important it is which of the Random/FIFO/LIFO strategies are being used.

Finally, it is important to note that although the analysis above uses the terminology introduced in Section 2 (e.g., gain buckets), the results are independent of the particular implementation, and apply in general to similar data structures are used in order to enable the incremental updating of variable gains.[6]

# 5 Discussion

In this paper, we evaluated several tie-breaking heuristics for selecting variables to flip in GSAT. Randomly generated formulas from a difficult distribution were used to empirically compare the performances of the strategies.

In the context of standard GSAT (without random walk), we found that randomized tie-breaking heuristics performed best, while the FIFO strategy did relatively poorly, and the LIFO strategy performed very poorly. It is interesting to compare our results with that of [4, 6], who concluded that "...there is nothing essential about randomness of picking in GSAT (although

[6]All efficient implementations of GSAT which we are aware of use a similar incremental updating framework.

fairness is important)...". The poor performance of LIFO (an unfair strategy) is compatible with these previous conclusions, and can be attributed to an excessive locality in its selection of variables. However, our results with FIFO and the randomized heuristics yield some new conclusions.

For smaller problems of 50-100 variables (the size of the problems studied in [4, 6]), the fair but non-random FIFO strategy is competitive with the Random strategy in GSAT without random walk. However, as the problem size increases, the performance of FIFO becomes significantly worse than that of Random. We believe that this is because variables are "stuck" in the FIFO queue too long; the fixed ordering of FIFO is too inflexible, and lacks the ability to exploit situations when a more local choice of variables is useful (i.e., when it is better to pick variables that are near the tail of the queue). This weakness of the FIFO strategy becomes increasingly significant as the problem sizes are scaled up, due to two reasons. First, as the number of variables increases, the size of the best gain bucket increases (Gent and Walsh [5] showed that the size of the best gain bucket scales linearly with the size of the problem). In addition, we showed in Section 4 that for smaller problem instances, the variable selection heuristic has a less significant impact than for large problem instances, because each variable flip has a more significant disruptive effect on the gain table structure for small problem instances than for large problem instances.

In contrast to the non-random strategies, the Random strategy offers the flexibility of choosing any variable from a gain bucket at any time, although the cumulative probability of choosing a variable as it is repeatedly offered for selection increases. This seems to be a desirable behavior which offers a balance between fairness and locality. An evaluation of some hybrid strategies which combined non-random (LIFO/FIFO) and Random strategies supports the importance of randomness. Significant improvements in performance can be seen as the amount of randomness is increased in these hybrid strategies.

Finally, the results of the comparison of tie-breaking heuristics in GSAT with random walk show that the context in which a variable selection strategy is evaluated is significant. Surprisingly, the FIFO strategy, which performed relatively poorly when used with GSAT without random walk, performed very well when used in the context of GSAT with random walk. Also, the LIFO strategy performs significantly better with random walk than without random walk. We attribute these improvements in the performance of the non-random strategies to the disruptive effect of the random walk on the gain table. That is, with a high enough probability of a random walk, the random flips cause enough movement in the gain table so that the variables are less prone to becoming stuck in a particular gain bucket, and the particular choice of variable selection heuristic (i.e., the choice between FIFO and RANDOM strategies) can become less important.

Thus, the main result of this paper is empirical evidence that randomness plays a much more significant role in variable selection than previously concluded by [4, 6], particularly for larger problem instances, and that fairness by itself is not a sufficient property for a successful strategy. Our analytical results offer some insight into why a fair, non-random strategy could yield results competitive with a random strategy for relatively small problem instances (such as those used in [4, 6]).

Given these results, we conclude that the use of the Random strategy, as originally proposed by [10], is indeed a reasonable choice. Furthermore, this work demonstrates the utility of revisiting empirical studies as increased computational resources become available – by scaling up the studies to study larger problem instances, it may be possible to gain new insights and obtain qualitatively different results, as was the case with this study.

## Acknowledgments

The research described in this paper was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Thanks to Rich Korf for helpful comments on a draft of this paper.

## References

[1] C.J. Alpert and A.B. Kahng. Recent directions in netlist partitioning: a survey. *Integration: the VLSI Journal*, 19:1–81, 1995.

[2] J.M. Crawford and L.D. Auton. Experimental results on the crossover point in random 3-sat. *Artificial Intelligence*, 81:31–57, 1996.

[3] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Computing Machinery*, 7:201–215, 1960.

[4] I.P. Gent and T. Walsh. The enigma of sat hill-climbing procedures. Research paper 605, Dept. of AI, University of Edinburgh, 1992.

[5] I.P. Gent and T. Walsh. An empirical analysis of search in gsat. *Journal of Artificial Intelligence Research*, 1:47–59, 1993.

[6] I.P. Gent and T. Walsh. Towards an understainding of hill-climbing procedures for sat. In *Proc. AAAI*, pages 28–33, 1993.

[7] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. In *Proc. AAAI*, pages 459–65, 1992.

[8] B. Selman and H. Kautz. Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *Proc. Intl. Joint Conf. Artificial Intelligence*, 1993.

[9] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Second DIMACS Challenge on Cliques, Coloring, and Satisfiability*, 1993.

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